Palms Vacation Properties

Group Num. 3

Aurora Hernandez

CMPS 3420

Fall 2018

Table of Contents

1 Fact-Finding, Information Gathering, and Conceptual Database Design

The objective of this document is to describe fact finding techniques, define methods used for gathering data and outline how to build a conceptual database design for Palms Vacation Properties. The database for our enterprise is carefully structured and organized based on specific requirements that are further detailed in our conceptual database design.

1.1 Fact-finding Techniques and Information Gathering

In creating an eloquent conceptual database design, we must first understand how to properly organize and collect data.Therefore, the target of this segment is to provide an introduction to our enterprise, give a description of fact finding techniques, and define a list of itemized descriptions of entity sets and relationships sets.

1.1.1 Introduction to Enterprise

Palms Vacation Properties is an enterprise headquarted in Bakersfield, California that operates online but serves cities all around the world. This enterprise allows people to rent and book vacation Properties for long term or short term periods. The Palms Vacation Properties range from houses, condos, boathouses, cottages and rooms. The purpose of Palms Vacation Properties is to serve as an intermediator and facilitate hospitatily services through peer-to-peer property rental agreements, meaning owners to guest and vice versa.

1.1.2 Description of Fact-Finding Techniques

The process of fact finding and information retrieval is lengthy and takes time. However, having accessability to various internet sites helps make our search less lengthy. The foundation of the database came from different online sources and from researching industries that handle hospitality services such as Airbnb and others.

Additionally, we surveyed property owners to find out what sort of information they would like to access in a database. The owners preferred to manage the listings on their own. They wanted to be able list their property at their own convinience, list their prices for nightly, weekly, and monthly stays depending on term of rental and seasons, manage their own descriptions, manage their available dates, list amenities, disclose cancellation policies, request deposits upfront and request monetary compesation for any damages.

On the other hand, users want to be able to create their own user accounts free-of-charge, add their personal information, view available properties in specific locations, make

and manage their reservations, send requests for special pricing or discounts, request specific dates in advance directly from owners, and be able to give property ratings.

1.1.3 Scope of the Database

The extent of Palms Vacation Properties database is to handle all organization, documentation, and information both professionally and securely. Simultaneously, representing a real world aspect known as *Miniworld* or *the universe of discourse (UoD)*.

Moreover, Palms Vacation Properties seeks to record only relevant information to carry out the necessary services to both owners and guests and dispose of any private data such as credit card information and other private data as soon all parties have been approved and any pending business between the two-parties has resolved. The changes to our miniworld is reflected in the next few sections which seeks to present a logical, concise and coherent collection of data for the purpose of creating a reliable database.

1.1.4 Description of Entity Sets and Relationships

This section's purpose is to briefly explain the contents and the conceptual definition of all entities and relationships. Each entity and relationships will be further defined in section 1.2, where appropriate contraints and specifics will be noted.

Entities

- **Owner**: Owner ID, First name, Middle name, Last name, Email

  An owner represents the owner of a one or more properties. An owner has first, middle and last name, email. Owner also registers his phone numbers, and operating addresses. Additionally, an owner offers a property for reservation and booking.

- **Guest**: Guest ID, First name, Middle name, Last name, Email

  A guest represents the person that makes booking one or more properties. A guest must register his phone numbers, and also specifies the address where he lives at. A guest reserves a property through a booking.

- **Property**: Property ID, Description, Rooms, Size, Amenities, Attractions, Views, Price

  A property offered for booking by a property owner. A booking selects one out of many properties. A property is offered by one or more owners.

- **Booking**: Booking ID, Daily Price, Booking Date, Checkin Date, Checkout Date, Booking Status, Confirmation Number

Booking allows guests to choose a checkin/checkout dates, check their confirmation number to verify the reservation, and once reservations are made it allows guests to lock in their daily price.

- **PropertyRating**: Property Rating ID, Rating, Review, Rating Date

  A review given by guests to rate and review their experiences for their bookings and the property owners.

- **Address**: Address ID, Street, City, Postal code, Country

  An address, can be for contact address, billing address, and/or property address

- **Phone**: Phone ID, Phone Number

  A contact's phone numbers. It also includes the phone type. For example, phone type can be a mobile phone, work phone, home work, and so forth

Relationships

- **Owner *Offers* Property** - Cardinality is Many to Many (N:M). Participation is Total - Total
- **Property *Includes* Address** - Cardinality is Many to Many (N:M). Participation is Partial - Total
- **Booking *Selects* Property** - Cardinality is One to Many (1:N). Participation is Total - Partial
- **Owner *Operates At* Address** - Cardinality is Many to Many (N:M). Participation is Total - Partial
- **Owner *Registers* Phone** - Cardinality is One to Many (1:N). Participation is Partial - Total
- **Guest *Registers* Phone** - Cardinality is One to Many (1:N). Participation is Partial - Total
- **Guest *Reserves* Booking** - Cardinality is One to Many (1:N). Participation is Total - Partial
- **Guest *Lives At* Address** - Cardinality is Many to Many (N:M). Participation is Total - Partial
- **Guest *Gives* PropertyRating** - Cardinality is One to Many (1:N). Participation is Partial - Total

- **PropertyRating *Rates* Property** - Cardinality is Many to One (N:1). Participation is Total - Partial
- **PropertyRating *Relates To* Booking** - Cardinality is One to Many (1:N). Participation is Total - Partial

1.1.5 User Groups, Data Views, and Operations

In most cases database designers interact with potential group of users and develop views of the the database. They use information from their interactions to extract data and begin processing data requirements for those specific groups. Each view of a user group is then analyzed and incorporated with the view of other user groups.

The idea is to take the views and connect them into our database design and store each logical data item in one place in the database, this is known as *data normalization* which makes our databse more fluid and saves storage space. The database design must eventually integrate all the different user groups, data views and operations in order to support all of the requirements of all the user groups.

1.2 Conceptual Database Design

Before we can dive into conceptual database design, we have to understand how to properly structure and store the data we have collected. The conceptual modeling of a database is very important phase because it is the phase where the foundation of the database queries and updates begin.

In the design of Palms Vacation Properties database we use the concepts of what is known as *Entity-Relationship (ER) model* which is a high-level conceptual data model. We use this model to further design our conceptual schemas for our database application.

In this section we discuss in detail entity type descriptions by giving it name(s) and relationship(s) description(s) of that entity with other entitity type(s). We further define relationship type descriptions, discuss any participation constraints and how each related entity type(s) are related. Then, we label and present a visual Entity Relationship (ER) Model that has entity types with attributes and relationship types which ties in our conceptual database design.

1.2.1 Entity Set Description

Entity: Owner

Description: An Owner entity represents an owner of a Property

Table 1: Owner

Attribute Name	**Owner ID**	**First Name**	**Middle Name**	**Last Name**	**Email**
Description	Primary Key	First name	Middle Name	Last Name	Email Address
Domain/Type	Integer	String	String	String	String
Value/Range	0 - Max Int	1-32 chars.	1-32 chars.	1-32 chars.	4-64 chars.
Unique	Yes	No	No	No	No
Null Allowed	No	No	Yes	No	No
Default Value	None	None	None	None	None
Single or Multivalue	Single	Single	Single	Single	Single
Simple or Composite	Simple	Simple	Simple	Simple	Simple

Entity: Guest

Description: A Guest entity represents a person that makes a reservation in Booking

Table 2: Guest

Attribute Name	**Guest ID**	**First Name**	**Middle Name**	**Last Name**	**Email**
Description	Primary Key	First name	Middle Name	Last Name	Email Address
Domain/Type	Integer	String	String	String	String
Value/Range	0 - Max Int	1-32 chars.	1-32 chars.	1-32 chars.	4-64 chars.
Unique	Yes	No	No	No	No
Null Allowed	No	No	Yes	No	No
Default Value	None	None	None	None	None
Single or Multivalue	Single	Single	Single	Single	Single
Simple or Composite	Simple	Simple	Simple	Simple	Simple

Entity: Property

Description: A Property entity represents a physical location of a specific property that is listed and that may be offered for reservation for guests

Table 3: Property

Attribute Name	Property ID	Description	Rooms	Size
Description	Primary Key	Prop. description	Number of Rooms	Square Feet
Domain/Type	Integer	String	String	Integer
Value/Range	0 - Max Int	64 - 5120 chars.	64 - 5120 chars.	1 - Max Int
Unique	Yes	No	No	No
Null Allowed	No	No	Yes	Yes
Default Value	None	None	None	None
Single or Multivalue	Single	Single	Single	Single
Simple or Composite	Simple	Simple	Simple	Simple

Table 3: Property - Continued

Attribute Name	Amenities	Attractions	Views	Price
Description	List of amenities	List of attractions	List of views	Price
Domain/Type	String	String	String	Numeric
Value/Range	Microwave, Roll-in bed, swimming pool, washer, dryer, and more	Monuments, Theme-parks, Museums, City Center, Library, and more	Ocean, Beach, Desert, Lake, Vineyard, City, Mountain, Nature	0-Max Numeric
Unique	No	No	No	No
Null Allowed	Yes	Yes	Yes	No
Default Value	None	None	None	None
Single or Multivalue	Multivalue	Multivalue	Multivalue	Single
Simple or Composite	Simple	Simple	Simple	Simple

Entity: Booking

Description: A Guest's record containing information about booking of a property. Provides the abstraction for reservation and booking, throughout the lifetime of a booking, such as booking status Pending, Reserved, Completed, Cancelled, and Void. Additionally, it contains the booking date, the checkin and check out dates, the Guest's confirmation number, and lastly the daily price quoted in case of a change in the duration of a booking.

Table 4: Booking

Attribute Name	**Booking ID**	**Daily Price**	**Booking Date**	**Checkin Date**
Description	Primary Key	Price per day	Date of booking	Date of checkin
Domain/Type	Integer	Numeric	Date	Date
Value/Range	0 - Max Int	0-Numeric Max	Today's date+	Today's date+
Unique	Yes	No	No	No
Null Allowed	No	No	Yes	No
Default Value	None	None	None	None
Single or Multivalue	Single	Single	Single	Single
Simple or Composite	Simple	Simple	Simple	Simple

Table 4: Booking - Continued

Attribute Name	**Checkout Date**	**Booking Status**	**Confirm. Number**
Description	Date of checkout	Status of booking	Confirmation Nr.
Domain/Type	Date	Enumerated	String
Value/Range	After checkin date	Pending, Reserved, Completd, Cancelled, Void	1-64 chars
Unique	No	No	Yes
Null Allowed	No	No	Yes
Default Value	None	Processing	None
Single or Multivalue	Single	Single	Single
Simple or Composite	Simple	Simple	Simple

Entity: PropertyRating

Description: A guest can choose to give a property owner, and a booking a rating and an optional review at the end of his or her stay. The guest rating consists of a rating score between 0 and 5. A brief review, and the date of the review.

Table 5: PropertyRating

Attribute Name	PropertyRating ID	Rating	Review	Rating Date
Description	Primary Key	Guest's Rating	Guest review	Date of rating
Domain/Type	Integer	Numeric	String	Date and Time
Value/Range	0 - Max Int	0-5	1-1024 chars.	Today's date+
Unique	Yes	No	No	No
Null Allowed	No	No	No	No
Default Value	None	None	None	today
Single or Multivalue	Single	Single	Single	Single
Simple or Composite	Simple	Simple	Simple	Simple

Entity: Address

Description: An address stores street, city, postal code (ZIP), and country. Used for storing the billing address of a Booking, in addition to a Contact's address.

Table 6: Address

Attribute Name	Address ID	Street	City	Postal Code	Country
Description	Primary Key	Street name	Name of city	Postal Code (ZIP)	Name of country
Domain/Type	Integer	String	String	String	String
Value/Range	0 - Max Int	1-64 chars	1-64 chars	1-16 chars	1-64 chars
Unique	Yes	No	No	No	No
Null Allowed	No	No	No	No	No
Default Value	None	None	None	None	None
Single or Multivalue	Single	Single	Single	Single	Single
Simple or Composite	Simple	Simple	Simple	Simple	Simple

Entity: Phone

Description: A phone number that a Contact may have associated with them. A Contact can have numerous phone numbers associated, and of different types such as Work Phone, Mobile Phone, Home Phone, and so forth.

Table 7: Phone

Attribute Name	**Phone ID**	**Phone Number**	**Phone Type**
Description	Primary Key	Phone Number	Type of phone nr.
Domain/Type	Integer	String	Enumeration
Value/Range	0 - Max Int	7-32 chars	Work, Home, Mobile, Other
Unique	Yes	No	No
Null Allowed	No	No	No
Default Value	None	None	No
Single or Multivalue	Single	Single	Single
Simple or Composite	Simple	Simple	Simple

1.2.2 Relationship Set Description

The following section shows the relationship details and the specifics of how entities are related.

Table 8: Owner *Offers* Property

Relationship	Owner **Offers** Property
Description	A Property must be owned by an Owner, and an Owner can not exist wihout a Property
Entity Set Involved	Owner, Property
Cardinality	Many to Many (N:M)
Descriptive Field(s)	Start Date, End Date
Participation Constraint	Owners offer one or more Property, and a Property must be offered by at least one Owner. Therefore, the participation constraint for Owner is total and for Property is Total

Table 9: Property *Includes* Address

Relationship	Property **Includes** Address
Description	A property includes one or more addresses
Entity Set Involved	Property, Address
Cardinality	Many to Many (N:M)
Descriptive Field(s)	None
Participation Constraint	Property must include an Address, and an address can contain one or more properties. However, an Address may exist without a property. Therefore, the participation constraint for property is Total and for Address is partial

Table 10: Booking *Selects* Property

Relationship	Booking **Selects** Property
Description	A booking must select a property
Entity Set Involved	Booking, Property
Cardinality	Many to One (N:1)
Descriptive Field(s)	None
Participation Constraint	Booking must select one Property. However, a Property must not necessarily be selected by a booking. Therefore, the participation constraint for Booking is total and for Property is partial

Table 11: Owner *Operates At* Address

Relationship	Owner **Operates At** Address
Description	A property includes one or more addresses
Entity Set Involved	Owner, Address
Cardinality	Many to Many (N:M)
Descriptive Field(s)	Start Date, End Date
Participation Constraint	An owner can operate at one or more addresses, and an address can be operated at by one or more property owners. A owner can not exist without at least one address. However, an Address may exist without a property. Therefore, the participation constraint for owner is partial and for Address is total

Table 12: Owner *Registers* Phone

Relationship	Owner **Registers** Phone
Description	An Owner can choose to register a Phone
Entity Set Involved	Owner, Phone
Cardinality	One to Many (1:N)
Descriptive Field(s)	Start Date, End Date
Participation Constraint	An owner can register one or more phone numbers. A Phone can not exist without being registered by an Owner. Therefore, the participation constraint for Owner is partial and Phone is partial

Table 13: Guest *Registers* Phone

Relationship	Guest **Registers** Phone
Description	An Guest can choose to register a Phone
Entity Set Involved	Guest, Phone
Cardinality	One to Many (1:N)
Descriptive Field(s)	Start Date, End Date
Participation Constraint	A Guest can register one or more Phone numbers. A phone can not exist without being registered by an Guest. Therefore, the participation constraint for Guest is partial and Phone is partial

Table 14: Guest *Reserves* Booking

Relationship	Guest **Reserves** Booking
Description	A Guest must reserve a Booking
Entity Set Involved	Guest, Booking
Cardinality	One to Many (1:N)
Descriptive Field(s)	Start Date, End Date
Participation Constraint	A Guest may reserve a Booking, and every booking must be reserved by a Guest. Therefore, the participation constraint for Guest is total and for Booking is total

Table 15: Guest *Lives At* Address

Relationship	Guest **Lives At** Address
Description	A property includes one or more addresses
Entity Set Involved	Guest, Address
Cardinality	Many to Many (N:M)
Descriptive Field(s)	Start Date, End Date
Participation Constraint	A Guest can live at one or more addresses, and an address can be lived at by one or more property owners. A owner can not exist without at least one address. However, an Address may exist without a property. Therefore, the participation constraint for owner is partial and for Address is total

Table 16: Guest *Gives* PropertyRating

Relationship	Guest **Gives** PropertyRating
Description	Gues may receive guest rating
Entity Set Involved	Guest, PropertyRating
Cardinality	One to Many (1:N)
Descriptive Field(s)	None
Participation Constraint	A Guest may give one or more PropertyRating. However, not all Guests must give a PropertyRating, but a PropertyRating can only be given by a Guest. Therefore, the participation constraint for Guest is partial and for GuestRating is Total

Table 17: PropertyRating *Rates* Property

Relationship	PropertyRating **Rates** Property
Description	An PropertyRating must rate a Property
Entity Set Involved	PropertyRating, Property
Cardinality	Many to One (N:1)
Descriptive Field(s)	None
Participation Constraint	A PropertyRating must rates a Property. A PropertyRating can not exist without rating a Property. Therefore, the participation constraint for PropertyRating is total, and for Property is partial

Table 18: PropertyRating *Relates To* Booking

Relationship	PropertyRating **Relates To** Booking
Description	PropertyRating must relate to a Booking
Entity Set Involved	PropertyRating, Booking
Cardinality	One to Many (1:N)
Descriptive Field(s)	None
Participation Constraint	An PropertyRating must relate to a Booking. A Booking can exist without being related to a PropertyRating. Therefore, the participation constraint for PropertyRating is total and for Property is partial

1.2.3 Related Entity Set

In this section we discuss related entity sets and relationships that are derived through a generalization and specialization process. There is concepts of *class* and *subclass* in the Entity Relationship (ER) model. Also, in the model we have concepts of *subclass* and *superclass* that includes related concepts such as *specialization* and *generalization* which is used to represent the collection of entities.

Specialization is the process of defining a set of *subclasses* of an entity type which is called the *superclass* of the specialization. As an example, we could consider the set of subclass to be the *owner* and *guest* to be a subclass of *contact* because this distinguishes *owner* and *guest* as two separate contacts of the entity.

Generalization is a process of abstraction which suppresses the differences among several entity types and *generalize* them into one *superclass* of which the original entity types are *subclasses*. We use the term generalization to define a generalized process for entity types from the given entity types. We can illustrate this by pointing out that *Phone Type*, *View Type* and *Property Type* are all generalized under *Property* superclass since they inherited some of the same attributes.

We can conclude that the *generalization* term is the inverse of *specialization*. There are two types of constraints that we address in our database schema. The first is the *participation constraint* which specifies the the existense of an entity depends on it being related to another entity via relationship type.

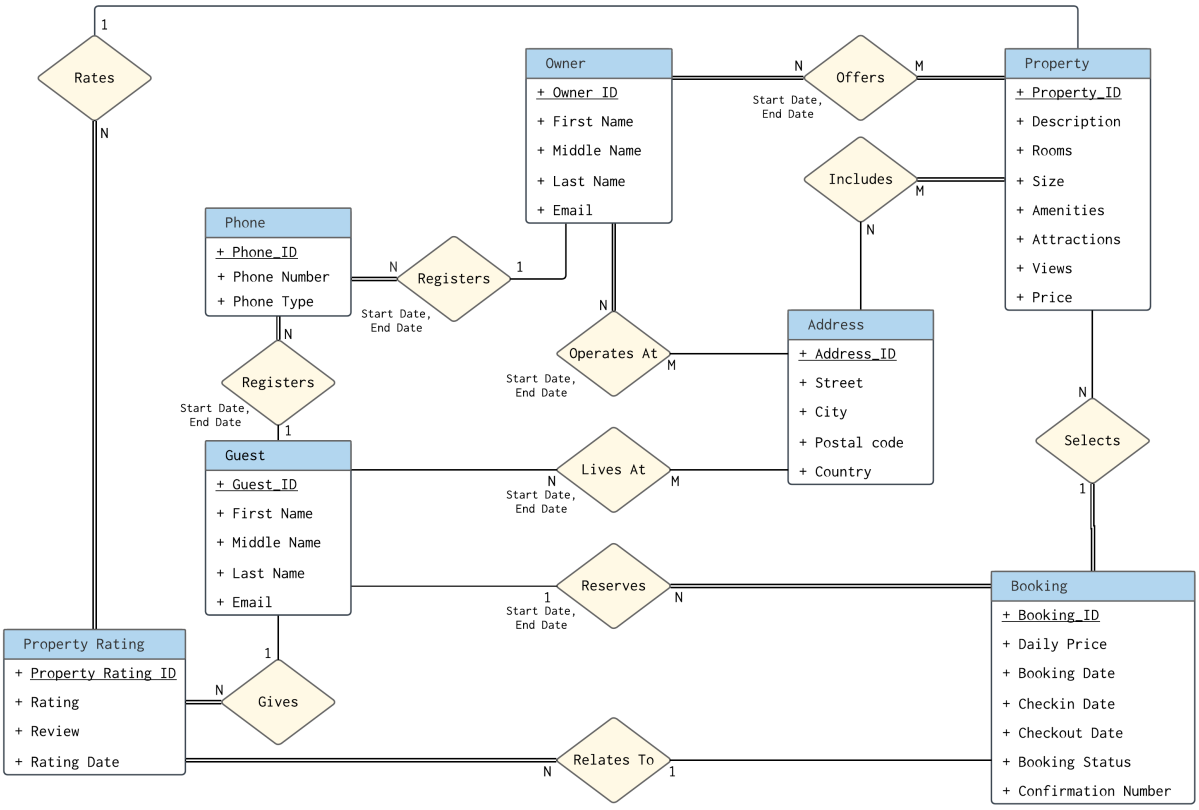
Additionally, this constraint specifies a minimum number of relationship instances that an entity can participate in and in some instances it is refer to as the *minium cardinality constraint*. The participation constraint has two types of constrains: partial and total. Total participation means that every entity in the *total set* an entity must be related to a specific entity via the relationship. Similarly, *partial* means that *some* part of the set an entity is related to some specific entity via a relationshp.

The second constraint we address is a constraint known as *disjoint constraint*. A disjoint

constraint is that which specifies that the subclasses of the specialization must be disjoint. This tells us that an entity can be a member of at least one of the subclasses of the specialization.

Also, if the subclasses have no disjoint constraint then their sets may have a member that is in one or more subclasses of specialization. Then,we have another constraint called *completeness*, which is also known as *totalness*. A total specialization constraint tells us that every entity in a superclass isa member of a minimum of one subclass in the specialization.

1.2.4 E-R Diagram



2 Phase 2: Conceptual Database and Logical Database

The second phase of this document is to convert our Entity Relationship (ER) Model database into a relational database. Our objective for this phase is to produce a conceptual schema for the database that is independent of a specific database management system. In order to have a clear conceptual database we must concentrate on specifying the properties of our data, without being concerned with storage and implementation details. This eventually enables us to create a good conceptual database design.

However, in order to have a good conceptual design we must take the next step in our database design which is to begin implementation of the database. This step is known as the *logical design* or *data model mapping*, this result is a database schema in the implementation data model of the database management system.

In addition to this, we re-examine our conceptual schema design and examine the data requirements resulting from our Phase 1. Then we outline our transaction and application design and produce a high-level specifications for those applications.

Furthermore, the next few sections will cover an array of information that explains how the conceptual database and logical database tie in together into Palms Vacation Rentals database. We use *conceptual database* to form envision and conceptualize our database while *logical database* is used to implement our database.

The first section of this covers the descriptions of both E-R model and relational model. Secondly, we re-discuss some major points of conceptual database and logical database. Afterwards, we introduce the methods of converting relationship types to relations and lastly we define some database constraints. The specifics of each section is discussed with much more detail in the subsections to follow.

2.1 E-R Model and Relational Model

This section gives a brief re-introduction to *Entity Relationship(ER) model* and an introduction to *Relational model*. The ER model describes our conceptual database design and the relational model describes the logical design of our Palms Vacation Properties database.

Additionally, we give a brief history of the models, discuss what the models are, outline some major features of the two models and analyze the purpose of the models. Then, we will compare the two different models, explore advantages vs disadvantages, and examine their differences and similarities.

2.1.1 Description of E-R Model and Relational Model

The *Entity-Relationship model* was introduced by Peter Chen and published in 1976 but some related work appears in the works of Schmidt and Swenson (1975), Wiederhold and

Elmasri(1979) and Senko (1975). The Entity-Relationship model is referred to as ER model for short. The ER model is a method for describing and creating conceptual database designs. The ER model allows database designers to represent interrelated objects. Furthermore, we use the ER model concepts for the conceptual design of Palms Vacation Properties database.

The ER model features and describes data such as *entities, relationships* and *attributes*. The purpose of the ER model is to help us define and design the database process and present database requirements for Palms Vacation Properties. Additionally, the ER model concepts help produce complex attributes, present methods for specifying the structural constraints on relationship types and convey conceptual ideas into a visual diagram that can be easily understood.

The *Relational model* was introduced by Ted Codd of IBM Research in 1970. The Relational model uses concepts of a mathematical relation and has theoretical basis in set theory and first-order predicate logic. The first commercial implementation of the relational model was available around the1980's.

The *Relational model* represents the database as a collection of *relations*. The basic *Relational data model* represents a database as an assemblage of tables where each table can be stored as a separate file. It is sort of a *flat* file of records–it is referred to as a *flat file* mostly because each record has a simple linear structure.

We can think of a relation as a *table* of value where each row in the table represents a collection of related data values. Similarly, a *row* represents a particular fact that may corresponding to a real-world entity or relationship. The table name and column have designated names and the names are used to help us interpret the meaning of the values of each row. The aforementioned is an informal definition to help us understand the relational model.

Formally, the *relational model* terminology is as follows: a row is called a *tuple*, a column header is refer to as an *attribute*, the table is called a *relational*, and the data type describing the types of values that can appear in each column is referred to by a *domain* of possible values. From here on, we formally refer to the terms discussed as–*tuple, attribute, relation* and *domain*.

The purpose of the Relational model is to provide a declarative method to specify data and queries. Additionally, database designers can specify what sort of information the database contains and what type of information they want to see and extract from it. We can let our database management system take care of structuring our data, storing the data and retrieve data from our queries.

2.1.2 Comparison of Two Different Models

There are some differences between the Entity Relationship Model and Relational Model. One of the main differences between the ER and Relational model is that relationship types

are not represented explicitly and instead they are represented by having two attributes. For example, if we have attribute A and attribute B attribute, A would be a primary key and attribute B would be a foreign key included in two relations S and T, while the two tuples S and T are related when they have the same value for A and B. In the relational schema we are able to create a separate relation for each multivalued attribute. This can cause confusion for some at times.

However, an advantage of the relational model is that its mathematically defined and structured. It uses *first-order predicate logic* and it is clearly grouped into realations.These relations provide a declarative method that helps specify data and queries. The relational model uses *Structured Query Language (SQL)* for its design, stream processing and management.

On the other hand, the ER model focuses on simplicity and readability of a database design. The ER model provides readability and simplicity because of the easy-to-read diagrams, ability to change the diagrams, present the diagrams to users and be able to understand what is going with the design process.

However, there are disadvantages in the ER model. One disadvantage is that there no standardirized definitions on how to create a database design. A single database design can have various different implementations in an ER model and one can have various solutions for it. This can cause chaos whithin a large group of users and/or database designers since one design can have many different outlooks. Not only that but also the ER model is not mathematically grounded and there is no language to structure it.

The following discusses some of the major differences:

ER Model	**Relational Model**
1:1 or 1:N relationship type	Foreign key (or *relationship* relation)
M:N relationship type	*Relationship* relation and *two* foreign keys
n-ary relationship type	*Relationship* relation and n foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Value set	Domain
Key attribute	Primary (or secondary) key

2.2 From Conceptual Database to Logical Database

Previously, we discussed the advantages and disavantages of both the *ER model* and *Relational model*. Now, we describe and provide detailed information concerning the conversion of our Palms Vacation Properties Conceptual database model to a logical database model.

In the first section we describe process of converting entity types to relations and the methods for converting. Secondly, we outline the conversion of relationship types to relations and the methods of the conversion. Thirdly, we specify database constraints and the purpose of those constraints. The sections are more detailed in the next few pages.

2.2.1 Converting Entity Types to Relations

In order to convert our *ER model* to *Relational model*, we need to begin by converting *entity types* to *relations*. To accomplish this, we create *relational schema* that all *entity types* represent.

The process of conversion is enlongated due to the complexity of the relational schema. The relational schema contains list of attributes with single-value domain. On the other hand, the complexities of the ER model lie in its composite and multi-value attributes, strong and weak entity tpes. This section helps clarify the complexities between *strong* entity types vs *weak* entity types, mapping *simple* and *composite* attributes, and lastly the mapping of *single* and *multi-value* attributes.

Converting of Strong Entity Types into Relations
Strong entity types, are also known as *regular entity types* which are labeled with a varible E and converted into one R in the relation schema. The relation R includes all of the simple simple attributes that are in E. We can choose one of the key attributes of E as the primary key for R. Then, if the key of E is a composite, then the set of simple attributes that form it will form the primary key of R (see **Simple and Composite Attributes** and **Single and Multi-value Attributes**). If multiplekeys are identited for E then the information detailing the attributes that form each addititional key is kept in order to specify unique keys of relation R.

Converting of Weak Entity Types into Relations
Weak entity types are labeled with the same name into relation R. In thi case, the weak entity types' partial key attributes of R and the placement of foreign key attributes which maps into R become the primary key for R. After this, we can begin to view R as a representation of a strong entity and we can use them as such. The relation R will then fully be mapped with each simple valued attribute of the weak entity valued or the composite attributes depending on the entity.

Mapping of Simple and Composite Attributes
The mapping of *simple* and *composite* attributes have two separate processes but are similar. The single attributes can be mapped directly to the relational R. However, the composite attributes, we can only add the simple attributes to separate it and map its simple components to our relation schema.

Mapping of Single and Multi-Value Attributres
The mapping of *single* and *multi-valued attributes* have separate processes. The *single-valued* attribute of an entity type is mapped dirrectly to our relation schema R. However, if we have a *multi-valued* attribute then we it is a bit more complex. Its complexities lie in that each multi-valued attribute of an entity type is represented with a new, separate relation of R. The old relation of R has to include an attribute corresponding to the new R plus the primary key attribute as a foreign key in R of the relation that represents the entity type or relationship type that has the new relation R as a multivalued attribute. In this case, if the multivalued attribute is composite, we can include its simple components.

2.2.2 Converting Rrelationship Types to Relations

When we explore the ER model, we can quickly notice that there are two main concepts: *entities* and *relationships*. On the other hand, the Relational model has one concept: *relation*. The ER model has a very visual and high level representation while the Relational model has to be shown in a relational schema. Moreover, this section covers a variety of similar topics on methods for converting such as:

- Relationship tpes with different cardinality: one-to-one, many-to-one and many to many relationships
- The representation of *'IsA'* (superclass and subclass) relationships along with *'HasA'* relationships
- Relationship types involving other relationships
- Recursive relationships involving other relationsips
- Relationships involving more than 2 entity types
- Relationships and categories or union types

Mapping Relationship Types with 1:1 Cardinality
When we have a relationship type R that has a 1:1 cardinality constraint in our ER schema then it can be identified as the relations of *R1* and *R2* that may be converted to the entity types in R. In order to approach this, we have three-separate possibilities as discussed here:

1. **Foreign Key Approach:**

   This approach of a foreign key of R2 is placed in R1, which contains the primary key of R2. If the primary key of R2 is made of multiple attributes, then all associated attributes will be added onto R1.

2. **Merged Relation Approach:**

   The merged relation approach of a 1:1 relatinship type is to merge two entity types and their ralationship into a signle realation. But this is ossible if both participants are total and the tables would have the same number of tuples.

3. **Cross Reference Approach:**

   The cross reference approach is for cross-referencing the primary keys of the two relations *R1* and *R2* and also known as *relationship relation approach*. In this case, each tupple in R is represented in a relationship instance that is related to *R1* and *R2*. The relation *R* includes the primary key attributes of *R1* and *R2* as foreign keys.

There are advantages and disadvantages to the three approaches mentioned above. The advantage of the *Foreign Key Approach* is that the amount of join operations is decreased but disadvantaged in that it can only be used when there is *total* participation. The*Merged Relation Approach* is not helpful nor useful in this case since in our ER schema we can combine two relations into one and this happens early in the conceptual design. The *Cross Reference Approach* is helpful if the the participanting entity types do not have *total* participation but the downside is that it increases the number of joins when we have to do query it.

Mapping Relationship Types with 1:N Cardinality

A relationship type describes the interaction between entity types *S1* and *S2* with a 1:N (one-to-many) cardinality constraint then we have to convert to *R1* and *R2*. After we do this, then we can relate *R1* to be related to multiple instance of *R2* and *R2* can be related to only once instance of *R1*. In mapping realtionship types with 1:N cardinality there are two-methods which are futher discussed here:

1. **Foreign Key Approach:**

   In the *foreign key Approach* is similar to 1:1 cardinality but different in that a foreign key from *R1* is place in *R2* which contains the primary key of *R1*. This makes the related side to the other entity type 1:N and the side of N cardinality gets the foreign key.

2. **Cross Reference approach:**

The *cross reference approach* is the third approach from 1:1 cardinality and quite the same. Here, we create a separate relation R whose attributes are the primary keys and foreign keys of *R1* and *R2*.The primary key of the newly created relation becomes the foreign key of the N cardinality.

The advantages and disadvantages are the same for 1:N and 1:1 relationship mappings as mentioned above: The advantage of the *Foreign Key Approach* is that the amount of join operations is decreased but disadvantaged in that it can only be used when there is *total* participation. The *Cross Reference Approach* is helpful if the the participanting entity types do not have *total* participation but the downside is that it increases the number of joins when we have to do query it.

Mapping Relationship Types with M:N Cardinality
For each relationship type R we can relate it to *S1* and *S2* with M:N (many-to-many) cardinality and converted to relations of *R1* and *R2*. Once we have done this, we can directly related every tuple of *R1* to have multiple relatable tuples of *R2*. This same description is applicable to *R2* which also has its tuple related to multiple relatable tuples of *R1*. The mapping relationship types with M:N cardinality has only one method for converting with is through the following:

- **Cross Reference Approach:**

  The *cross reference approach* is carried out by creating a relationship R to represent the relatinship type. This relatinship contains the primary keys for *R1* and *R2* as foreign keys. In the scenario that the primay keys of *R1* or *R2* have multiple attributes then the those attributes become a *combination* that forms the primary key of R.

Mapping Superclasses and Subclasses for the 'IsA' Relationship
The mapping of *superclasses* and *subclasses* for the *'IsA'* relationship occurs when the entity types are *disjoint* subclasses of a superclass. There are three ways to approach this:

1. **Multiple Relations- superclass and subclass:**

   The *multiple relations- superclass and subclass* approach occurs when a superclass entity is given a relation of super*S1* and it will contain attributes of *S1* Then a subclass entity is given a relation of sub*S2* which contains the attributes of *S2* along with the primary key of the superclass relation as a foreign key attribute.

2. **Multiple relations - subclass only:**

   In the *multipel relations- subclass only* approach; the relations exist for only subclasses. The subclass entities are given their own relations. The subclass has the union of the

attributes of the attributes form subclass entity type and superclass entity type. In case we have overlapping subclasses then all the corresponding subclass relations have to share the same primary key values.

3. **Single relation with one type attribute:**

   The *single relation with one type attribute* approah occurs when a single realtion R is created and it holds the the union of the attributes from superclass entity types and all the subclass entity types.

There are disadvantages and advantages for the three approaches aforementioned. The *Multiple relations - superclass and subclass* approach is useful mostly since it works well all the superclass and subclass realation *disjoint* of *IsA* but the downfall is that it requires more join operations when we query it. The *Multiple relations - subclass only* approach is helpful in that it decreases the number of joins as compared to the first approah. The disadvantage is that it works only if the superclass and subclass have total participantion. The *Single relation with one type attribute* approach requires the least number of jon operations since it should be used when we have a large amount of related subclasses but it can eventually become more complex if we have large numbers of subclasses.

Mapping Superclasses and Subclasses for the 'HasA' Relationship

The *mapping superclasses and subclasses for the "IsA" relationship* occurs when there is *overlapping* between sublcass entity types, this tells us that there can be multiple subclases. For this approach, there are thow methods:

1. **Multiple relations - superclass and subclass:**

   The *multiple relatoins superclass and subclass* is handled in the same manner as *'IsA'* method.

2. **Single relation with multiple type attributes:**

   The *single relation with multiple type attributes* approach a relation R is created that has the union attributes of superclass and all the attributes from subclass. This also contains *boolean type attribute* that distinguishes whether a tuple belongs to subclass *S2*

Furthermore, both of the methods have their adavantages and disadvantages. The *multiple relations - superclass and subclass* has the same advantages and disadvantages as *'IsA'* relationship. The *single relation with one type attribute* requires the least number of Furthermore, both of the methods for *join* operations but its disadvantage lies in that it can become difficult if we have many subclasses. However, the method should only be used among related subclases for more efficient implementation.

Mapping Relationship Types to Other Relationship Types
When it comes to *mapping relaionship types to other relatinship types*, one must create a single primary key for the relationship type. After this, we can map the relationship between the relationship types using **foreign key approach** or the **cross-reference approach**, but this depends on the cardinality of the relatinships being mapped.

Mapping Recursive Relationship Types
The *mapping recursive relationship types* deals with when an entity type that is represented by R is related to itself. There are two methods to represent this:

1. **Foreign Key Approach:** In the *foreign key approach* a relation is created from entity type R. The relation created holds the foreign key attribute that is referenced to the primary key.

2. **Cross Reference Approach:** In the *cross reference approach* we create a new relation. This newly created relation represents the recursive relationship. The new relation contains two foreign keys that reference the primary key of R. Then, the foreign keys become associated with the relation that is combined and those keys become the primary key of the new created recursive relation.

There are disadvantages and adavantages for both approaches The *foreign key approach* requires less *join* operations but a relation that is not recursive holds a Null value which is not efficient and wastes space. The *cross reference approach* is a bit more efficient but needs more *join* operations when we query it.

Mapping Relationship Types for Ternary Entity Types or Greater
The *mapping relationship types for ternanry entity types or greater* we have a relationshp type that is connected with more than two entity types. In this case, the relatinship type is presented as its own relationship relation of R. Every entity type is converted into relations and the relaionship type R holds the primary keys of the relation as foreign keys.

Mapping of Union Types
The *mapping of union types* occurs when a subclass belongs to multiple superclasses. The relations that are associated with different superclass entities have different primary keys. The new key attribute is known as a *surrogate key*. In the case that we have multiple entities are superclasses of the same subclass entity then it will share the same value for the surrogate key since the relaion tuples will correspond to the superclass entities.

2.2.3 Database Constraints

Throughout the conception and design of Palms Vacation Properties database there have been many important features. One of the important features lies in the *constraints* of the

relational database. In order to validate the relational database and make it meaningful, we need *constraints* to ensure that all the data being stored is relevent.

Moreover, in this section we describe the purpose of the following constraints and how a database management system (DBMS) enforces the constraints:

- **Entity Integrity Constraints**
  The *entity Integrity constraint* tells us that a primary key value can not have a value of NULL. The primary key value can not be NULL because each tuple must be uniquely identified accordingly.

- **Primary Key Constraints**
  The *primary key constraints*, is the key whose values are used to *identity* tuples in the relation. The values of the primary key are not the same for any two typles in the relation and the constraint helps ensure that.

- **Unique Key Constraints**
  The *unique key constraints* specifies wether NUll values are or are not permitted. For example, if every Guest tuple must have a valid, non-NULL value for the 'Name' attribute then the 'Name' of the Guest is constrained to unique. This constraint ensures that the given attribute will be unique for every tuple in the relation.

- **Referential Constraints**
  The *referential constraints* is among two relations and helps us to maintain the consistenty between two relations. This constraint ensures the foreign key does not reference an entity that does not exist. Also, the condition for a foreign key specifies constraint between two relation schemas *R1* and *R2*.

- **Check Constraints**
  The *check constraints* are checks that may automatically check to make sure a set of conditions are met before sending out any changes to a relation. The types of checks that may be ran include checking for non NULL values and the likes.

- **Business Rules** When dealing wiht *business rules constraints*, in large applications, it means that some constraints may have to be checked and/or updated by programs, and at sometimes at the time of the data entry. For example, a *property* apartment address may be listed or lack an identifying number or letter such as *'10A'* instead of *'10B'*, then the DBMS can not detect this error because both inputs are valid values for the *address*. This type of error has to be corrected manually if a *Guest* goes to the property and the address can not be found, then the data base must be updated.

2.3 Convert Entity Relationship Model into a Relational Database

In this section, we will convert our ER Conceptual datatabse into a Relational/Logical Database for Palms Vacation Properties. First, we will discuss the *relation schema* for our database then sample data of relation. After we discuss the methods then we will sample and design queries for our database using *Relational Algebra Expressions*, *Tuple Relational Calculus* and *Domain Relational Calculus* to showcase the functions of our database.

2.3.1 Relational Schema for Logical Database

Table 1: Relation Schema: Owner

Attribute	Type
oid	Integer, (0 - Max Integer)
firstname	varchar(1,32)
middlename	varchar(1,32)
lastname	varchar(1,32)
email	varchar(4,64)

- **Candidate Keys:**

  oid

- **Primary Key / Entity Integrity Constraint:**

  oid must be a non-null and a unique value

- **Referential Integrity Constraint:**

- **Unique Constraint:**

- **Business Constraint:**

  Firstname, lastname and email must not be null

- **Derivation From Entity and Relationship Types:**

  Owner

- **Summary:**

  Derived from the Owner entity. Also connected to other entities such as Property, Address and Phone

Table 2: Relation Schema: Guest

Attribute	Type
gid	Integer, (0 - Max Integer)
firstname	varchar(1,32)
middlename	varchar(1,32)
lastname	varchar(1,32)
email	varchar(4,64)

- **Candidate Keys:**

  gid

- **Primary Key / Entity Integrity Constraint:**

  gid must be a non-null and a unique value

- **Referential Integrity Constraint:**

- **Unique Constraint:**

- **Business Constraint:**

  Firstname, lastname and email must not be null

- **Derivation From Entity and Relationship Types:**

  Owner

- **Summary:**

  Derived from the Guest entity.

Table 3: Relation Schema: Property

Attribute	Type
pid	Integer, (0 - Max Integer)
description	varchar(64,5120)
rooms	Integer, (0 - Max Integer)
size	Integer, (0 - Max Integer)
price	Decimal

- **Candidate Keys:**

  pid

- **Primary Key / Entity Integrity Constraint:**

  pid must be a non-null and a unique value

- **Referential Integrity Constraint:**

- **Unique Constraint:**

- **Business Constraint:**

  The description must be a non-null and non-empty value. Size should also be a positive integer value.

- **Derivation From Entity and Relationship Types:**

  Propety

- **Summary:**

  Derived from the Property entity

Table 4: Relation Schema: Amenity

Attribute	Type
pid	Integer, (0 - Max Integer)
name	varchar(32)

- **Candidate Keys:**

  pid

- **Primary Key / Entity Integrity Constraint:**

  pid must be a non-null and a unique value

- **Referential Integrity Constraint:**

  pid must be a reference to valid Property primary key

- **Unique Constraint:**

- **Business Constraint:**

  Amenity name value must be unique for every pid; no duplicate entries

- **Derivation From Entity and Relationship Types:**

  Property amenities multivalue attribute

- **Summary:**

  Derived from the Property amenities multivalue attribute

Table 5: Relation Schema: Attraction

Attribute	Type
pid	Integer, (0 - Max Integer)
name	varchar(32)

- **Candidate Keys:**

  pid

- **Primary Key / Entity Integrity Constraint:**

  pid must be a non-null and a unique value

- **Referential Integrity Constraint:**

  pid must be a reference to valid Property primary key

- **Unique Constraint:**

- **Business Constraint:**

  Attraction name value must be unique for every pid; no duplicate entries

- **Derivation From Entity and Relationship Types:**

  Property attractions multivalue attribute

- **Summary:**

  Derived from the Property attractions multivalue attribute

Table 6: Relation Schema: View

Attribute	Type
pid	Integer, (0 - Max Integer)
name	varchar(32)

- **Candidate Keys:**

  pid

- **Primary Key / Entity Integrity Constraint:**

  pid must be a non-null and a unique value

- **Referential Integrity Constraint:**

  pid must be a reference to valid Property primary key

- **Unique Constraint:**

- **Business Constraint:**

  View name value must be unique for every pid; no duplicate entries

- **Derivation From Entity and Relationship Types:**

  Property views multivalue attribute

- **Summary:**

  Derived from the Property views multivalue attribute

Table 7: Relation Schema: Booking

Attribute	Type
bid	Integer, (0 - Max Integer)
daily_price	Decimal
booking_date	Date and Time
cin	Date and Time
cout	Date and Time
booking_status	Enum: Pending, Reserved, Completed, Cancelled, Void
confirm_nr	varchar(64)
pid	Integer, (0 - Max Integer)
gid	Integer, (0 - Max Integer)

- **Candidate Keys:**

  bid, pid, gid, confirm_nr

- **Primary Key / Entity Integrity Constraint:**

  bid, pid, and gid must be non-null unique values

- **Referential Integrity Constraint:**

  pid and gid must be a reference to a valid Booking primary key

- **Unique Constraint:**

  confirm_nr must be a unique, if not a nul-value

- **Business Constraint:**

  Checkin date (cin) must be a valid date date and time before check out date (cout) and time. Confirm_nr must be set if booking_status is set to reserved or completed.

- **Derivation From Entity and Relationship Types:**

  Booking, Reserves, Selects

- **Summary:**

  Derived from the Booking entity. Pid and gid foreign keys derived from the Guest 'Reserves' Booking and Booking 'Selects' Property entity relationship.

Table 8: Relation Schema: property_rating

Attribute	Type
prid	Integer, (0 - Max Integer)
rating	Integer, [0-5]
review	varchar(1,1024)
rating_date	Date and Time
pid	Integer, (0 - Max Integer)
bid	Integer, (0 - Max Integer)
gid	Integer, (0 - Max Integer)

- **Candidate Keys:**

  prid

- **Primary Key / Entity Integrity Constraint:**

  prid must be a non-null and a unique value

- **Referential Integrity Constraint:**

- **Unique Constraint:**

  should be unique as multiple reviews for the same booking should not be allowed

- **Business Constraint:**

- **Derivation From Entity and Relationship Types:**

  PropertyRating, Rates, Gives, Relates To

- **Summary:**

  Derived from the PropertyRating entity. Pid, bid, and gid are derived from Property Rating 'Rates' Property, Guest 'Gives' Property Rating, and Property Rating 'Relates To' Booking entity relationships.

Table 9: Relation Schema: Address

Attribute	Type
aid	Integer, (0 - Max Integer)
street	varchar(1,64)
city	varchar(1,64)
postal_code	varchar(1,16)
country	varchar(1,64)

- **Candidate Keys:**

  aid

- **Primary Key / Entity Integrity Constraint:**

  aid must be a non-null and a unique value

- **Referential Integrity Constraint:**

- **Unique Constraint:**

- **Business Constraint:**

  street, city, postal_code and country must be non-null values

- **Derivation From Entity and Relationship Types:**

  Address

- **Summary:**

  Derived from the Address entity

Table 10: Relation Schema: Phone

Attribute	Type
phid	Integer, (0 - Max Integer)
phone_number	varchar(7,32)
phone_type	Enumerated: Work, Home, Mobile, Other
gid	Integer, (0 - Max Integer)
oid	Integer, (0 - Max Integer)

- **Candidate Keys:**

  phid, gid, oid

- **Primary Key / Entity Integrity Constraint:**

  phid must be a non-null and a unique value

- **Referential Integrity Constraint:**

  gid and oid must be a reference to valid Guest and Owner primary keys

- **Unique Constraint:**

- **Business Constraint:**

  phone_number must be provided.

- **Derivation From Entity and Relationship Types:**

  Phone, Owner 'Registers' Phone and Guest 'Registers' Phone

- **Summary:**

  Derived from the Phone entity. Also models Owner 'Registers' Phone and Guest 'Registers' Phone entity relationships.

Table 11: Relation Schema: Offers

Attribute	Type
oid	Integer, (0 - Max Integer)
pid	Integer, (0 - Max Integer)
start_date	Date and Time
end_date	Date and Time

- **Candidate Keys:**

  oid, pid

- **Primary Key / Entity Integrity Constraint:**

  oid and pid must be a non-null

- **Referential Integrity Constraint:**

  oid and pid must refer to valid Owner and Property primary keys

- **Unique Constraint:**

  oid and pid must be unique within the same time period of start_date and end_date

- **Business Constraint:**

- **Derivation From Entity and Relationship Types:**

  Offers

- **Summary:**

  Derived from the Offers entity relationship

Table 12: Relation Schema: Includes

Attribute	Type
pid	Integer, (0 - Max Integer)
aid	Integer, (0 - Max Integer)

- **Candidate Keys:**

  pid, aid

- **Primary Key / Entity Integrity Constraint:**

  pid and aid must be a non-null

- **Referential Integrity Constraint:**

  pid and aid must refer to valid Property and Address primary keys

- **Unique Constraint:**

  pid and aid must be a unique combination

- **Business Constraint:**

- **Derivation From Entity and Relationship Types:**

  Includes

- **Summary:**

  Derived from the Includes entity relationship

Table 13: Relation Schema: Selects

Attribute	Type
bid	Integer, (0 - Max Integer)
pid	Integer, (0 - Max Integer)

- **Candidate Keys:**

  bid, pid

- **Primary Key / Entity Integrity Constraint:**

  bid and pid must be a non-null

- **Referential Integrity Constraint:**

  bid and pid must refer to valid Booking and Property primary keys

- **Unique Constraint:**

  bid and pid must be a unique combination; no double booking the same property

- **Business Constraint:**

- **Derivation From Entity and Relationship Types:**

  Selects

- **Summary:**

  Derived from the Selects entity relationship

Table 14: Relation Schema: LivesAt

Attribute	Type
gid	Integer, (0 - Max Integer)
aid	Integer, (0 - Max Integer)
start_date	Time and Date
end_date	Time and Date

- **Candidate Keys:**

  gid, aid

- **Primary Key / Entity Integrity Constraint:**

  gid and aid must be a non-null

- **Referential Integrity Constraint:**

  gid and aid must refer to valid Guest and Address primary keys

- **Unique Constraint:**

  gid, aid must be unique within the same period of start_date and end_date

- **Business Constraint:**

  start_date must not be null.

- **Derivation From Entity and Relationship Types:**

  LivesAt

- **Summary:**

  Derived from the LivesAt entity

Table 15: Relation Schema: OperatesAt

Attribute	Type
oid	Integer, (0 - Max Integer)
pid	Integer, (0 - Max Integer)
start_date	Time and Date
end_date	Time and Date

- **Candidate Keys:**

  oid, pid

- **Primary Key / Entity Integrity Constraint:**

  oid and pid must be a non-null

- **Referential Integrity Constraint:**

  oid and pid must refer to valid Owner and Property primary keys

- **Unique Constraint:**

  oid, pid must be unique within the same period of start_date and end_date

- **Business Constraint:**

- **Derivation From Entity and Relationship Types:**

  OperatesAt

- **Summary:**

  Derived from the OperatesAt entity relationship

2.3.2 Sample Data of Relation

Table 1: Owner Example Data Table

oid	firstname	middlename	lastname	email
1	Dacia	Jules	Stidson	jstidson0@yale.edu
2	Nikolos		Whittier	nwhittier1@multiply.com
3	Harlan	Noellyn	Lieb	nlieb2@hostgator.com
4	Elisa	Garland	Scrowton	gscrowton3@gmpg.org
5	Gilburt	Cherye	Kennsley	ckennsley4@cam.ac.uk
6	Spike		Swithenby	tswithenby5@buzzfeed.com
7	Jean	Cherilynn	Jeandel	cjeandel6@si.edu
8	Fiorenze	Erek	Greve	egreve7@google.es
9	Corie	Celina	Hansod	chansod8@mashable.com
10	Brenden	Lief	Kosiada	lkosiada9@berkeley.edu
11	Weylin	Sorcha	Keppe	skeppea@gravatar.com
12	Brena	Moritz	Newgrosh	mnewgroshb@topsy.com
13	Manya	Coriss	Gonoude	cgonoudec@gizmodo.com
14	Tessi		Gullivent	jgulliventd@unicef.org
15	Culley	Quillan	Gout	qgoute@adobe.com
16	Aeriel		Denmead	rdenmeadf@a8.net
17	Vale		Gadaud	kgadaudg@godaddy.com
18	Jonis	Hieronymus	Carillo	hcarilloh@usnews.com
19	Flory	Roley	Brightey	rbrighteyi@cargocollective.com
20	Anestassia	Isadora	Agett	iagettj@tamu.edu
21	Melita	Margit	McFeate	mmcfeatek@mapquest.com
22	Raeann	Aindrea	Hartridge	ahartridgel@acquirethisname.com
23	Derril		Van der Son	avandersonm@tinypic.com
24	Bernice		Fleckness	cflecknessn@google.com.br
25	Maddy	Irene	Dufore	iduforeo@cnn.com
26	Alyda	Becki	Musgrave	bmusgravep@goo.ne.jp
27	Talia	Axe	Bathow	abathowq@microsoft.com
28	Talia	Andriette	Fernely	afernelyr@nbcnews.com
29	Kara-lynn	Joaquin	Penton	jpentons@pen.io

oid	firstname	middlename	lastname	email
30	Othello	Trix	De Carlo	tdecarlot@youtube.com
31	Orelle		Muttock	mmuttocku@51.la
32	Louella		Sieghart	asieghartv@fda.gov
33	Jerrilyn	Simone	Salthouse	ssalthousew@cargocollective.com
34	Deerdre	Tedmund	Mc Combe	tmccombex@webeden.co.uk
35	Patrica	Aeriell	Mallabar	amallabary@phpbb.com
36	Dre		Karoly	okarolyz@mapquest.com
37	Karlie		Ivamy	rivamy10@gmpg.org
38	Marabel		Bellson	tbellson11@istockphoto.com
39	Tudor		Tasch	rtasch12@sfgate.com
40	Vivianna	Lurlene	Droghan	ldroghan13@ed.gov
41	Talia	Baily	Flattman	bflattman14@msn.com
42	Aarika	Brien	Eloy	beloy15@discuz.net
43	Effie	Noble	Shorie	nshorie16@gov.uk
44	Ignaz	Brannon	McSharry	bmcsharry17@cnbc.com
45	Tonya	Iggie	Domanski	idomanski18@stanford.edu
46	Osbourn		Allmann	eallmann19@privacy.gov.au
47	Sigvard	Milena	Scriver	mscriver1a@columbia.edu
48	Valle	Fidela	Fairbridge	ffairbridge1b@surveymonkey.com
49	Pierette	Davidde	Culshaw	dculshaw1c@example.com
50	Chandler	Lishe	Fewless	lfewless1d@1688.com
51	Kellen	Filippa	Tribble	ftribble1e@yale.edu
52	Shay	Chev	Deville	cdeville1f@jugem.jp
53	Ginevra		Ehlerding	eehlerding1g@exblog.jp
54	Darelle	Conroy	Stanney	cstanney1h@woothemes.com
55	Ban	Allison	Harradine	aharradine1i@ycombinator.com
56	Artemis	Lian	Comello	lcomello1j@about.me
57	Emmalyn	Clarissa	Hurche	churche1k@domainmarket.com
58	Thia	Shaina	Massingberd	smassingberd1l@webmd.com
59	Luce	Fancy	Cicullo	fcicullo1m@vkontakte.ru

oid	firstname	middlename	lastname	email
60	Kiley	Sophia	Stennard	sstennard1n@jiathis.com
61	Jerrie	Elfie	Newall	enewall1o@godaddy.com
62	Donaugh	Thornton	Geleman	tgeleman1p@moonfruit.com
63	Berget	Faunie	Comello	fcomello1q@behance.net
64	Antonella	Rowney	Tootell	rtootell1r@jugem.jp
65	Rock	Thorin	Gavagan	tgavagan1s@networksolutions.com
66	Jillian	Bartel	Piddocke	bpiddocke1t@ucla.edu
67	Barb	Ricky	Bussell	rbussell1u@timesonline.co.uk
68	Carole		Blint	pblint1v@wikispaces.com
69	Ericka	Conan	Raatz	craatz1w@seesaa.net
70	Alysia	Legra	Jeayes	ljeayes1x@newyorker.com
71	Jodie	Auroora	Testo	atesto1y@wikia.com
72	Alexander	Domingo	Freeman	dfreeman1z@chicagotribune.com
73	Essie		Scarasbrick	mscarasbrick20@apache.org
74	Jasen		Frear	gfrear21@people.com.cn
75	Kippy		Mostin	lmostin22@wufoo.com
76	Taite	Prissie	Beckensall	pbeckensall23@angelfire.com
77	Maryjane	Rafael	Blannin	rblannin24@deliciousdays.com
78	Rori	Riki	Vaskov	rvaskov25@biblegateway.com
79	Gifford		Ricoald	tricoald26@howstuffworks.com
80	Anna-maria		Bowler	jbowler27@dion.ne.jp
81	Christiane		Thirst	athirst28@hostgator.com
82	Ermentrude	Jud	Schankel	jschankel29@chron.com
83	Albert	Michael	Sparshott	msparshott2a@spiegel.de
84	Jedidiah	Marice	Whilde	mwhilde2b@prlog.org
85	Kelly	Townsend	Spincks	tspincks2c@archive.org
86	Tiffany	Tarah	Lines	tlines2d@ask.com
87	Bartholomew		Duthy	dduthy2e@de.vu
88	Debi	Candi	Gilburt	cgilburt2f@mayoclinic.com
89	Walsh	Ron	Farrah	rfarrah2g@discuz.net

oid	firstname	middlename	lastname	email
90	Brittney	Jessi	Treweke	jtreweke2h@etsy.com
91	Nathanael	Deane	Bage	dbage2i@squidoo.com
92	Hulda		Paynter	dpaynter2j@yellowbook.com
93	Philbert	Addia	McKimmie	amckimmie2k@homestead.com
94	Cordie	Dagny	Pounds	dpounds2l@latimes.com
95	Bride		Mergue	dmergue2m@webnode.com
96	Adriane	Barbi	Mazey	bmazey2n@nih.gov
97	Galvin	Jorrie	Rens	jrens2o@t.co
98	Ag	Mariel	Teare	mteare2p@wiley.com
99	Perkin	Jerry	Turbat	jturbat2q@businesswire.com
100	Salome	Archie	Kinset	akinset2r@cnbc.com

Table 2: Guest Example Data Table

oid	firstname	middlename	lastname	email
1	Laurent	Viv	Kingsbury	vkingsbury0@youtu.be
2	Rosa	Boniface	Punton	bpunton1@accuweather.com
3	Berti	Kelsy	Kybert	kkybert2@amazon.co.jp
4	Jan		Hing	bhing3@mysql.com
5	Joana	Cordi	Talks	ctalks4@about.me
6	Valdemar	Ana	Fears	afears5@e-recht24.de
7	Newton	Donni	Franklen	dfranklen6@elegantthemes.com
8	Mufinella		Grey	agrey7@slashdot.org
9	Shawn	Wilone	Pape	wpape8@hp.com
10	Butch	Ginni	Meagh	gmeagh9@addtoany.com
11	Tiphanie	Alvan	Gerssam	agerssama@dailymotion.com
12	Say	Jack	Pavkovic	jpavkovicb@newsvine.com
13	Turner	Alejandrina	Sarl	asarlc@oaic.gov.au
14	Stavro	Linc	Farren	lfarrend@t.co
15	Jorgan	Rosabelle	Lode	rlodee@mozilla.com
16	Brooks		Blunsum	gblunsumf@fc2.com
17	Jesus	Bordie	Sidnell	bsidnellg@newyorker.com
18	Isobel	Antoine	Bodicum	abodicumh@skyrock.com
19	Yancy	Bell	Allflatt	ballflatti@dot.gov
20	Janel	Zsa zsa	Kirkman	zkirkmanj@china.com.cn
21	Janet	Jon	Birtwhistle	jbirtwhistlek@quantcast.com
22	Vince		Wray	swrayl@ocn.ne.jp
23	Augusto	Parnell	John	pjohnm@sciencedaily.com
24	Daniela		Dowdeswell	ldowdeswelln@jimdo.com
25	Cyril	Griffy	Yard	gyardo@technorati.com
26	Candide	Griselda	Battelle	gbattellep@t-online.de
27	Roch	Lorry	Amner	lamnerq@naver.com
28	Kerry	Reider	Carswell	rcarswellr@sogou.com
29	Wallache	Merell	Allett	malletts@issuu.com

oid	firstname	middlename	lastname	email
30	Amby	Miltie	Lambden	mlambdent@google.com
31	Hyacinth	Fenelia	Breedy	fbreedyu@chicagotribune.com
32	Haleigh	Aldrich	Claffey	aclaffeyv@vimeo.com
33	Bone		Keary	dkearyw@gizmodo.com
34	Jacinda	Hastie	Fitzharris	hfitzharrisx@liveinternet.ru
35	Ervin	Ginger	Muzzullo	gmuzzulloy@vkontakte.ru
36	Leland	Bridie	Toth	btothz@constantcontact.com
37	Cara	Aluin	Petruk	apetruk10@fc2.com
38	Catherina		Korpolak	akorpolak11@dell.com
39	Loree		Crudginton	kcrudginton12@last.fm
40	Tripp		Sellars	rsellars13@webs.com
41	Frans	Mattheus	Swatridge	mswatridge14@ox.ac.uk
42	Roxie	Ulrike	Joron	ujoron15@nyu.edu
43	Eddi	Sheri	Mounch	smounch16@google.ru
44	Wandis	Anjanette	Frise	afrise17@sogou.com
45	Darby		Noden	znoden18@marriott.com
46	Leopold	Mitchell	Meikle	mmeikle19@technorati.com
47	Eldridge	Athene	Gillman	agillman1a@tinyurl.com
48	Miguel	Eudora	Dearell	edearell1b@friendfeed.com
49	Travers	Mozes	Fauning	mfauning1c@networkadvertising.org
50	Adella	Thornton	Redmond	tredmond1d@shareasale.com
51	Sheri	Garret	Gyorffy	ggyorffy1e@china.com.cn
52	Uta		Norkutt	dnorkutt1f@phpbb.com
53	Patty	Ayn	Aronstein	aaronstein1g@blogger.com
54	Lancelot		Eyton	deyton1h@geocities.jp
55	Clarisse		Habbert	rhabbert1i@umich.edu
56	Vittoria	Rodolfo	Ajsik	rajsik1j@virginia.edu
57	Marylin	Smith	Tracy	stracy1k@google.it
58	Kliment	Koo	Gasquoine	kgasquoine1l@yellowbook.com
59	Lars	Teriann	Ivett	tivett1m@pcworld.com

oid	firstname	middlename	lastname	email
60	Ogdon	Corette	Maultby	cmaultby1n@hud.gov
61	Clarissa		Urridge	kurridge1o@google.de
62	Randie	Edita	McCoveney	emccoveney1p@webnode.com
63	Wallace	Ferd	Sproat	fsproat1q@flavors.me
64	Tanhya		McCullum	bmccullum1r@indiegogo.com
65	Jerrilee	Carmencita	Ketley	cketley1s@free.fr
66	Immanuel	Pooh	Baskeyfied	pbaskeyfied1t@slashdot.org
67	Hermon		Medlin	emedlin1u@livejournal.com
68	Kettie	Brnaba	Mulvany	bmulvany1v@google.com.hk
69	Bette		Canape	ecanape1w@gnu.org
70	Brook	Gaston	Crusham	gcrusham1x@diigo.com
71	Clair	Tallie	Dell Casa	tdellcasa1y@slate.com
72	Valle	Therine	Jarret	tjarret1z@statcounter.com
73	Brynn		Penella	lpenella20@webs.com
74	Sammie		Mauser	rmauser21@ucsd.edu
75	Ken	Julianna	Crosscombe	jcrosscombe22@printfriendly.com
76	John		Smith	j.smith.coolio@arizona.edu
77	Fayina	Allyson	Minchin	aminchin24@purevolume.com
78	Rustin	Dacy	McAndie	dmcandie25@skype.com
79	Tanney		Rosbrough	rrosbrough26@paypal.com
80	Dacie		Binder	abinder27@merriam-webster.com
81	Thea	Hamid	Bruni	hbruni28@microsoft.com
82	Gypsy	Giusto	Gibard	ggibard29@mayoclinic.com
83	Caryn	Allistir	Bannon	abannon2a@meetup.com
84	Sallie		Lemerie	alemerie2b@goodreads.com
85	Nichol	Loydie	Arniz	larniz2c@hatena.ne.jp
86	Felipe		Gonnel	kgonnel2d@uiuc.edu
87	Emile	Erica	Dayes	edayes2e@dell.com
88	Oralee	Eldridge	Scade	escade2f@home.pl
89	Karee	Barry	O' Mulderrig	bomulderrig2g@example.com

oid	firstname	middlename	lastname	email
90	Kaycee		Purkess	hpurkess2h@i2i.jp
91	Wallas	Quentin	Brou	qbrou2i@chron.com
92	Bronson	Pierrette	Bein	pbein2j@blinklist.com
93	Phylis		Snuggs	asnuggs2k@bloomberg.com
94	Murvyn	Amara	Buckingham	abuckingham2l@geocities.jp
95	Gardener	Ashby	Kiff	akiff2m@hubpages.com
96	Cosette	Yoshiko	Piotrowski	ypiotrowski2n@imgur.com
97	Vance		Whitlaw	kwhitlaw2o@joomla.org
98	Ferdinande	Karleen	Seligson	kseligson2p@twitter.com
99	Ynes	Myrtice	Osbourne	mosbourne2q@engadget.com
100	Ricca		Ilden	milden2r@go.com

Table 3: Property Example Data Table

pid	description	rooms	size	price
1	turpis enim	2	1132	236.18
2	molestie lorem quisque	2	2556	267.41
3	ante ipsum	4	963	244.59
4	ultrices aliquet maecenas	1	1773	324.21
5	sociis natoque	4	2388	164.35
6	cursus id	3	658	167.84
7	eget tincidunt eget		2312	93.07
8	eget eros elementum	2	2744	156.3
9	congue etiam	10	2230	260.61
10	volutpat convallis morbi		2086	348.61
11	interdum venenatis	8	1252	264.63
12	nisl nunc nisl	9	1182	227.72
13	eleifend quam	5	803	78.22
14	amet turpis elementum	10	794	1096.77
15	dapibus dolor vel	2	1533	167.76
16	morbi quis tortor		2462	184.92
17	tristique tortor eu	5	955	340.61
18	quis orci	9	568	192.47
19	in est	5	1800	244.87
20	est donec	4	2284	335.63
21	odio consequat varius	5	2988	292.94
22	commodo placerat praesent	7	977	305.79
23	non velit nec	10		114.72
24	vestibulum vestibulum	1	367	214.74
25	interdum venenatis	9	2545	246.44
26	suscipit a feugiat			150.18
27	lectus in	5	2344	258.7
28	ligula pellentesque ultrices	4	1633	224.28
29	urna ut tellus	9	1972	135.94

pid	description	rooms	size	price
30	id mauris		2551	291.31
31	vestibulum sit	7	1974	296.09
32	mollis molestie lorem	7	2422	164.67
33	elit sodales scelerisque	5	817	283.58
34	tristique est et	7	2725	66.09
35	quis odio consequat	10	258	261.64
36	justo lacinia	3	2032	166.77
37	sit amet turpis	9	2556	169.24
38	posuere cubilia curae	3	2054	275.89
39	lorem ipsum dolor	10	2869	285.9
40	imperdiet et commodo	6	1833	319.16
41	justo sit	5	231	281.63
42	phasellus in	3	2615	123.72
43	nibh quisque		2073	138.61
44	elit proin interdum	7	1887	305.44
45	primis in	10	542	266.84
46	lacinia nisi	6	2622	249.08
47	proin eu	10	984	44.93
48	sapien quis libero	5	2428	131.21
49	purus phasellus in	9	2887	65.88
50	nam congue risus		913	86.9
51	elit sodales	1		176.54
52	ipsum dolor sit	2		77.91
53	neque vestibulum	7	965	69.86
54	donec pharetra magna	6	1046	290.7
55	mattis nibh	10	1763	239.67
56	pulvinar nulla pede	7		294.14
57	sapien ut	7	890	174.48
58	pede venenatis	1	1022	139.58
59	nunc vestibulum ante		790	219.04

pid	description	rooms	size	price
60	amet eleifend pede	1		207.29
61	lobortis est		557	41.65
62	elementum nullam varius	4	2835	233.24
63	sapien dignissim vestibulum	3	2110	1181.86
64	morbi odio odio		438	173.01
65	mattis odio	9	2845	116.14
66	aliquet ultrices erat	9	2587	119.71
67	luctus cum	2	297	126.65
68	orci luctus et	3	249	314.0
69	varius integer ac	8	865	198.34
70	in porttitor pede	2	557	122.04
71	elementum nullam varius	9	811	261.95
72	integer non velit	8	584	294.71
73	donec ut	8		277.26
74	sapien iaculis congue		2656	246.69
75	diam vitae quam	3	2764	242.96
76	sodales scelerisque mauris	8	2010	1315.8
77	odio donec vitae		500	139.11
78	integer pede	6	2293	87.44
79	at turpis a	7	1703	103.35
80	aliquet at feugiat	9	667	203.1
81	et ultrices	6	2936	263.14
82	interdum mauris	4	551	146.41
83	erat quisque	5	854	81.27
84	ac est		1398	95.03
85	consequat lectus in		1231	277.25
86	tortor quis	4	1687	269.52
87	mi sit amet	10	2725	214.32
88	luctus et ultrices	2	1808	261.16
89	orci luctus	10		263.81

pid	description	rooms	size	price
90	scelerisque quam turpis		1530	204.3
91	morbi vel lectus	10		346.93
92	rutrum at lorem	10		148.22
93	rutrum neque	4	647	331.89
94	quis tortor id	2	2219	173.02
95	lobortis convallis tortor	9	312	333.46
96	consequat dui nec	1	1759	206.85
97	augue a	7	1154	201.17
98	sit amet nulla	9		127.83
99	vulputate nonummy	6	2951	191.79
100	nibh ligula nec	1	793	270.62

Table 4: Amenity Example Data Table

pid	name
25	Washer
1	Roll-in Bed
4	Swimming Pool
9	Roll-in Bed
18	Dryer
8	Swimming Pool
20	Roll-in Bed
8	Roll-in Bed
13	Roll-in Bed
8	Dryer
24	Dryer
4	Dryer
23	Washer
1	Swimming Pool
21	Washer
21	Roll-in Bed
5	Swimming Pool
14	Swimming Pool
20	Washer
11	Roll-in Bed
16	Washer
15	Microwave
7	Washer
19	Roll-in Bed
4	Washer
22	Roll-in Bed
13	Dryer
10	Dryer
22	Swimming Pool

pid	name
20	Swimming Pool
13	Washer
24	Washer
6	Dryer
15	Roll-in Bed
1	Dryer
10	Washer
8	Washer
15	Washer
22	Washer
10	Microwave
18	Swimming Pool
4	Roll-in Bed
10	Roll-in Bed
9	Microwave
6	Washer
3	Microwave
22	Microwave
22	Dryer
19	Dryer
20	Microwave

Table 5: Attraction Example Data Table

pid	name
17	City Center
15	Monuments
4	Theme-parks
20	Library
13	Theme-parks
19	Monuments
18	Monuments
23	Monuments
8	Museums
25	Museums
2	Monuments
3	Theme-parks
4	City Center
8	Monuments
16	City Center
24	City Center
1	Monuments
10	Museums
24	Theme-parks
24	Museums
14	Theme-parks
15	Museums
7	City Center
24	Library
22	Library
18	Theme-parks
24	Monuments
4	Museums
21	City Center

pid	name
11	Library
4	Monuments
10	Library
11	City Center
5	Theme-parks
22	Museums
19	Theme-parks
12	Monuments
15	City Center
21	Museums
6	Library
2	Museums
10	Theme-parks
7	Theme-parks
21	Monuments
19	City Center
9	Library
3	City Center
6	Museums
14	Museums
16	Library

Table 6: View Example Data Table

pid	name
8	Beach
13	Mountain
25	Desert
6	Vineyard
47	Beach
13	Vineyard
47	City
9	Beach
14	Lake
14	Nature
19	Nature
21	City
4	Desert
11	Ocean
21	Vineyard
55	City
25	Lake
17	Lake
20	Mountain
24	Nature
9	Vineyard
10	Vineyard
15	Nature
22	Vineyard
11	Mountain
67	City
3	Vineyard
70	City
2	City

pid	name
8	Vineyard
17	Desert
17	Mountain
2	Ocean
15	City
25	Ocean
70	Vineyard
12	City
13	Lake
25	Mountain
11	City
15	Ocean
5	Desert
12	Beach
25	Vineyard
96	City
13	Nature
10	Desert
97	City
21	Mountain
18	Mountain

Table 7: Booking Example Data Table

bid	daily_price	booking_date	cout	cout	booking_status	confirm_nr
1	286.12	2018-05-07	2018-10-24 15:00:00	2018-10-28 12:00:00	Canceled	
2	139.19	2018-10-21	2018-10-24 15:00:00	2018-10-29 12:00:00	Pending	
3	174.13	2018-04-12	2018-11-06 15:00:00	2018-11-08 12:00:00	Void	
4	200.82	2018-08-07	2018-09-29 15:00:00	2018-10-04 12:00:00	Pending	
5	153.47	2018-06-29	2018-09-28 15:00:00	2018-09-30 12:00:00	Completed	59e819b2
6	60.64	2018-07-01	2018-10-19 15:00:00	2018-10-25 12:00:00	Void	
7	129.1	2018-06-15	2018-10-28 15:00:00	2018-11-06 12:00:00	Completed	930d2ea6
8	225.39	2018-09-17	2018-10-02 15:00:00	2018-10-11 12:00:00	Void	
9	103.98	2018-03-02	2018-09-22 15:00:00	2018-09-30 12:00:00	Reserved	
10	263.19	2018-08-08	2018-09-21 15:00:00	2018-09-27 12:00:00	Completed	bcbaffec
11	286.76	2018-05-26	2018-10-09 15:00:00	2018-10-14 12:00:00	Pending	
12	335.89	2018-06-30	2018-11-11 15:00:00	2018-11-13 12:00:00	Reserved	
13	328.78	2018-05-13	2018-10-06 15:00:00	2018-10-13 12:00:00	Void	
14	188.9	2018-06-03	2018-10-01 15:00:00	2018-10-07 12:00:00	Canceled	
15	149.92	2018-04-21	2018-10-11 15:00:00	2018-10-13 12:00:00	Completed	b771aa80
16	226.13	2018-04-18	2018-10-20 15:00:00	2018-10-30 12:00:00	Reserved	
17	304.88	2018-07-30	2018-09-19 15:00:00	2018-09-20 12:00:00	Reserved	
18	316.97	2018-04-02	2018-10-12 15:00:00	2018-10-15 12:00:00	Void	
19	259.79	2018-04-21	2018-10-12 15:00:00	2018-10-14 12:00:00	Void	
20	1222.5	2018-05-24	2018-10-13 15:00:00	2018-10-18 12:00:00	Pending	
21	82.97	2018-06-12	2018-09-12 15:00:00	2018-09-15 12:00:00	Reserved	
22	215.6	2018-03-29	2018-09-30 15:00:00	2018-10-06 12:00:00	Completed	fdf70f9b
23	114.05	2018-05-06	2018-11-01 15:00:00	2018-11-10 12:00:00	Reserved	
24	49.91	2018-04-08	2018-10-12 15:00:00	2018-10-13 12:00:00	Completed	f7e96a27
25	97.66	2018-06-27	2018-10-08 15:00:00	2018-10-11 12:00:00	Reserved	
26	164.79	2018-06-17	2018-09-28 15:00:00	2018-10-07 12:00:00	Completed	fe4d3ed5
27	328.16	2018-02-13	2018-09-26 15:00:00	2018-10-01 12:00:00	Void	
28	336.46	2018-02-10	2018-10-04 15:00:00	2018-10-05 12:00:00	Completed	97ed550b
29	290.12	2018-03-26	2018-10-27 15:00:00	2018-10-28 12:00:00	Completed	87836d26

bid	daily_price	booking_date	cout	cout	booking_status	confirm_nr
30	272.02	2018-07-22	2018-11-09 15:00:00	2018-11-13 12:00:00	Pending	
31	137.36	2018-04-27	2018-10-06 15:00:00	2018-10-09 12:00:00	Canceled	
32	279.75	2018-03-16	2018-10-03 15:00:00	2018-10-05 12:00:00	Reserved	
33	283.67	2018-05-22	2018-10-26 15:00:00	2018-11-04 12:00:00	Reserved	
34	163.53	2018-05-18	2018-09-17 15:00:00	2018-09-21 12:00:00	Pending	
35	1328.18	2018-01-08	2018-11-04 15:00:00	2018-11-12 12:00:00	Completed	ede837ad
36	91.98	2018-03-28	2018-09-29 15:00:00	2018-10-05 12:00:00	Reserved	
37	61.46	2018-08-22	2018-10-23 15:00:00	2018-10-28 12:00:00	Reserved	
38	187.2	2018-04-30	2018-10-14 15:00:00	2018-10-22 12:00:00	Completed	9582b7ce
39	371.27	2018-02-02	2018-09-21 15:00:00	2018-09-27 12:00:00	Reserved	
40	345.02	2018-07-05	2018-10-31 15:00:00	2018-11-08 12:00:00	Pending	
41	391	2018-05-14	2018-10-06 15:00:00	2018-10-15 12:00:00	Canceled	
42	53.98	2018-05-04	2018-10-03 15:00:00	2018-10-09 12:00:00	Reserved	
43	1225.18	2018-08-26	2018-11-02 15:00:00	2018-11-10 12:00:00	Void	
44	166.97	2018-09-17	2018-09-23 15:00:00	2018-09-28 12:00:00	Void	
45	52.49	2018-06-12	2018-09-18 15:00:00	2018-09-22 12:00:00	Canceled	
46	59.13	2018-06-13	2018-09-14 15:00:00	2018-09-18 12:00:00	Pending	
47	71.49	2018-01-21	2018-10-15 15:00:00	2018-10-23 12:00:00	Completed	7fc5bf94
48	272.55	2018-06-04	2018-11-04 15:00:00	2018-11-05 12:00:00	Void	
49	86.09	2018-06-06	2018-09-27 15:00:00	2018-10-05 12:00:00	Pending	
50	202.96	2018-06-29	2018-11-04 15:00:00	2018-11-05 12:00:00	Reserved	
51	189.79	2018-05-01	2018-09-27 15:00:00	2018-10-04 12:00:00	Canceled	
52	71.9	2018-05-07	2018-10-28 15:00:00	2018-11-04 12:00:00	Completed	dabfa618
53	1111	2018-09-13	2018-10-18 15:00:00	2018-10-22 12:00:00	Completed	57532c4c
54	108.65	2018-07-13	2018-11-04 15:00:00	2018-11-11 12:00:00	Pending	
55	373.3	2018-05-06	2018-10-17 15:00:00	2018-10-24 12:00:00	Void	
56	67.56	2018-07-10	2018-09-28 15:00:00	2018-10-04 12:00:00	Void	
57	323	2018-05-03	2018-10-22 15:00:00	2018-10-31 12:00:00	Completed	c5a46f10
58	50.13	2018-09-28	2018-11-03 15:00:00	2018-11-04 12:00:00	Canceled	
59	355.92	2018-04-06	2018-10-23 15:00:00	2018-10-25 12:00:00	Void	

bid	daily_price	booking_date	cout	cout	booking_status	confirm_nr
60	254.94	2018-07-09	2018-09-20 15:00:00	2018-09-21 12:00:00	Pending	
61	251.06	2018-01-28	2018-10-22 15:00:00	2018-10-23 12:00:00	Pending	
62	230.48	2018-07-23	2018-09-13 15:00:00	2018-09-18 12:00:00	Pending	
63	81.25	2018-09-08	2018-09-27 15:00:00	2018-10-01 12:00:00	Void	
64	143.57	2018-09-09	2018-10-23 15:00:00	2018-10-27 12:00:00	Canceled	
65	381.09	2018-05-06	2018-10-06 15:00:00	2018-10-09 12:00:00	Void	
66	54.67	2018-06-20	2018-10-23 15:00:00	2018-10-28 12:00:00	Void	
67	372.49	2018-04-12	2018-11-07 15:00:00	2018-11-11 12:00:00	Pending	
68	225.41	2018-09-19	2018-10-06 15:00:00	2018-10-15 12:00:00	Canceled	
69	133.99	2018-04-25	2018-09-13 15:00:00	2018-09-14 12:00:00	Pending	
70	132.37	2018-06-17	2018-10-21 15:00:00	2018-10-22 12:00:00	Reserved	
71	47.17	2018-05-10	2018-11-03 15:00:00	2018-11-10 12:00:00	Completed	898d792b
72	60.89	2018-07-15	2018-11-02 15:00:00	2018-11-04 12:00:00	Pending	
73	213.8	2018-03-21	2018-10-07 15:00:00	2018-10-15 12:00:00	Canceled	
74	79.77	2018-05-26	2018-10-27 15:00:00	2018-10-31 12:00:00	Void	
75	353.68	2018-08-22	2018-10-22 15:00:00	2018-10-30 12:00:00	Completed	e3d61ddf
76	312.29	2018-08-15	2018-09-17 15:00:00	2018-09-21 12:00:00	Canceled	
77	259.98	2018-08-03	2018-09-26 15:00:00	2018-09-28 12:00:00	Void	
78	353.04	2018-02-07	2018-09-25 15:00:00	2018-09-26 12:00:00	Completed	48aaa41f
79	84.61	2018-05-17	2018-11-06 15:00:00	2018-11-08 12:00:00	Void	
80	117.4	2018-09-28	2018-10-31 15:00:00	2018-11-02 12:00:00	Pending	
81	335.67	2018-06-02	2018-10-28 15:00:00	2018-10-29 12:00:00	Canceled	
82	173.6	2018-04-01	2018-09-17 15:00:00	2018-09-26 12:00:00	Canceled	
83	1189.65	2018-01-16	2018-10-25 15:00:00	2018-10-31 12:00:00	Canceled	
84	99.4	2018-09-02	2018-09-13 15:00:00	2018-09-17 12:00:00	Canceled	
85	296.64	2018-01-30	2018-09-30 15:00:00	2018-10-04 12:00:00	Completed	6ecd554b
86	372.72	2018-03-31	2018-10-27 15:00:00	2018-10-29 12:00:00	Reserved	
87	163.33	2018-09-01	2018-11-03 15:00:00	2018-11-08 12:00:00	Reserved	
88	376.19	2018-02-21	2018-11-01 15:00:00	2018-11-08 12:00:00	Reserved	
89	141.96	2018-04-09	2018-09-27 15:00:00	2018-10-03 12:00:00	Void	

bid	daily_price	booking_date	cout	cout	booking_status	confirm_nr
90	32.19	2018-01-15	2018-10-08 15:00:00	2018-10-11 12:00:00	Void	
91	58.44	2018-08-13	2018-10-14 15:00:00	2018-10-21 12:00:00	Pending	
92	379.53	2018-03-16	2018-10-03 15:00:00	2018-10-10 12:00:00	Completed	ecfc52ec
93	37.08	2018-01-09	2018-09-28 15:00:00	2018-10-07 12:00:00	Pending	
94	213.02	2018-06-05	2018-10-21 15:00:00	2018-10-29 12:00:00	Reserved	
95	87.88	2018-03-23	2018-10-20 15:00:00	2018-10-22 12:00:00	Pending	
96	225.9	2018-02-01	2018-10-26 15:00:00	2018-10-30 12:00:00	Completed	fbc19155
97	343.79	2018-08-14	2018-11-06 15:00:00	2018-11-10 12:00:00	Void	
98	115.8	2018-08-12	2018-09-20 15:00:00	2018-09-28 12:00:00	Pending	
99	170.66	2018-09-05	2018-09-24 15:00:00	2018-10-02 12:00:00	Completed	3051728c
100	328.82	2018-09-16	2018-11-04 15:00:00	2018-11-05 12:00:00	Reserved	

Table 8: Property_Rating Example Data Table

prid	rating	review	rating_date	pid	bid	gid
1	1	pulvinar sed	2018-09-07	60	8	95
2	5		2018-01-26	49	91	91
3	5		2018-02-11	38	17	23
4	4		2017-11-11	21	34	76
5	4	urna	2018-02-02	44	4	70
6	5	cursus	2018-04-05	96	98	97
7	4	molestie lorem	2018-08-24	15	90	25
8	3		2018-10-04	72	96	47
9	4	et ultrices	2018-06-20	42	74	43
10	4		2017-11-17	21	68	8
11	5		2018-03-25	23	76	44
12	2	eget eleifend	2018-06-05	42	95	66
13	3	in	2018-05-16	16	36	4
14	4		2018-05-17	56	19	30
15	5		2018-01-21	55	2	63
16	4		2017-11-11	83	51	82
17	3		2018-01-27	91	89	83
18	3	non	2017-12-29	56	53	45
19	2		2018-02-24	33	8	75
20	4		2018-05-26	65	73	21
21	2	aliquam lacus	2018-03-11	87	13	59
22	4		2018-07-25	7	96	88
23	2	rutrum rutrum	2018-10-02	29	33	27
24	2	vel	2017-11-15	84	78	54
25	4		2018-02-06	36	27	22
26	5		2018-03-24	77	92	56
27	4	nisi eu	2018-04-24	76	100	6
28	4	in	2018-09-22	88	83	36
29	4	dui	2018-04-03	85	25	13

prid	rating	review	rating_date	pid	bid	gid
30	4		2018-05-09	87	31	84
31	4		2018-04-25	87	61	70
32	5		2018-01-21	33	65	87
33	2	vestibulum	2018-04-26	42	56	3
34	1		2018-02-08	52	73	32
35	3		2018-04-05	87	66	73
36	3	integer ac	2017-11-03	75	70	89
37	3		2017-11-24	70	81	84
38	4		2018-08-15	18	68	59
39	4	nulla nisl	2018-04-26	49	82	36
40	2		2018-07-26	15	23	68
41	4		2017-11-19	9	36	21
42	4		2017-12-29	3	33	50
43	2	a nibh	2018-07-22	87	96	36
44	2	vitae nisl	2017-10-31	81	65	71
45	4		2018-08-14	46	90	9
46	2		2018-07-11	39	72	35
47	4	sit	2018-06-09	21	77	62
48	4		2018-01-12	7	63	60
49	4	vivamus	2017-10-22	90	65	76
50	1		2018-03-06	33	78	94
51	2	in est	2018-02-27	37	54	75
52	5	rhoncus sed	2018-09-11	78	7	73
53	1		2017-11-12	19	34	88
54	4	donec odio	2018-07-07	8	19	69
55	3		2017-12-13	6	2	63
56	3	ac leo	2018-08-04	80	93	4
57	2		2018-05-13	52	42	9
58	2	donec posuere	2018-09-04	68	67	75
59	4		2018-09-05	27	63	64

prid	rating	review	rating_date	pid	bid	gid
60	4	enim	2018-05-25	80	69	75
61	4	feugiat et	2018-09-24	43	71	21
62	2		2018-09-11	61	73	20
63	2		2018-04-23	58	80	4
64	4		2017-12-30	43	29	25
65	2	dui	2018-05-18	4	34	84
66	4	metus	2018-08-31	66	3	69
67	4	est congue	2018-05-01	98	37	28
68	1		2018-08-29	15	44	11
69	4	est	2018-10-18	13	67	81
70	3	felis donec	2018-05-30	51	97	28
71	4	auctor gravida	2017-12-04	61	50	86
72	3	nulla ut	2018-07-29	57	79	41
73	1		2018-04-09	70	20	5
74	2		2018-03-29	19	83	12
75	1	semper	2018-07-15	42	62	55
76	5	orci vehicula	2017-12-13	34	100	21
77	3	in lectus	2018-08-29	28	82	79
78	1	at nibh	2018-06-23	34	11	6
79	4	lobortis sapien	2018-03-02	1	75	18
80	3	lacus	2017-10-30	17	21	40
81	5		2017-10-08	46	30	50
82	3		2018-10-23	12	1	15
83	4		2018-05-18	80	91	54
84	3	in sapien	2018-08-07	90	62	45
85	1	et ultrices	2018-04-25	68	84	36
86	4		2017-10-24	80	9	11
87	5	lacinia	2018-08-27	94	92	64
88	3		2017-10-06	40	21	68
89	4		2018-04-19	5	88	99

prid	rating	review	rating_date	pid	bid	gid
90	1	convallis nunc	2018-04-17	58	63	4
91	2	urna ut	2018-04-22	67	78	30
92	4	maecenas ut	2018-01-05	52	82	26
93	4	mattis	2018-02-08	72	30	2
94	1	lobortis vel	2018-01-16	26	56	37
95	4	nec	2018-02-19	10	27	20
96	1	in felis	2017-10-25	96	7	92
97	5		2017-12-03	4	25	27
98	4		2018-07-13	56	93	46
99	4	nunc rhoncus	2017-12-14	54	55	46
100	4	interdum	2018-08-28	18	60	69

Table 9: Address Example Data Table

aid	street	city	postal_code	country
1	9 Gerald Drive	San Juan	56030	Mexico
2	5931 Grim Lane	Santa Maria	51009	Mexico
3	21 Calypso Parkway	Virginia Beach	23459	United States
4	00485 Shopko Road	Shreveport	71151	United States
5	99273 Lunder Center	Wilkes Barre	18763	United States
6	6 Huxley Alley	Reston	20195	United States
7	360 Claremont Drive	Sainte-Thérèse	J7G	Canada
8	90590 Blue Bill Park Way	Altona	K1W	Canada
9	6 Northport Lane	Labrador City	A2V	Canada
10	945 Hovde Circle	Baie-Saint-Paul	G3Z	Canada
11	00668 Mockingbird Street	Tyler	75799	United States
12	094 Butterfield Alley	Emiliano Zapata	32883	Mexico
13	Elm Street	Las Vegas	89135	United States
14	3140 Spaight Circle	La Huerta	53338	Mexico
15	3656 Mockingbird Place	Bloomington	47405	United States
16	59816 Colorado Terrace	Kansas City	66112	United States
17	553 Di Loreto Avenue	Middleton	LE16	United Kingdom
18	64955 Schiller Center	Lubbock	79491	United States
19	5 Homewood Avenue	Dallas	75205	United States
20	6 Hayes Plaza	Atlanta	31165	United States
21	134 Lukken Drive	Norfolk	23504	United States
22	538 Marquette Avenue	Sacramento	94250	United States
23	4 Blackbird Drive	San Cristobal	36800	Mexico
24	4 Rutledge Junction	5 de Mayo	87785	Mexico
25	40350 Farwell Alley	Renfrew	K7V	Canada
26	62123 Portage Point	Milwaukee	53234	United States
27	2509 Marcy Circle	Revelstoke	H9K	Canada
28	65 Old Shore Plaza	Guadalupe Victoria	87086	Mexico
29	0716 Elm Street	Paterson	07505	United States

aid	street	city	postal_code	country
30	86487 Mayfield Park	Magrath	J7G	Canada
31	6 Briar Crest Place	Lubbock	79405	United States
32	1 Oakridge Junction	Newton	IV1	United Kingdom
33	76 Bayside Center	San Francisco	95710	United States
34	095 Crowley Hill	El Mirador	56564	Mexico
35	85885 Eggendart Trail	Killam	A0A	Canada
36	99 Tennyson Lane	Sacramento	95852	United States
37	65566 Eliot Center	Montgomery	36195	United States
38	1331 International Terrace	El Calvario	68213	Mexico
39	1222 Banding Pass	San Antonio	78220	United States
40	35886 Lakeland Place	Pueblo	81010	United States
41	11 Becker Place	Venustiano Carranza	93848	Mexico
42	61859 Northwestern Alley	San Isidro	60284	Mexico
43	7 Montana Parkway	Benito Juarez	91716	Mexico
44	1722 Division Court	Benito Juarez	41800	Mexico
45	03687 Ridge Oak Court	Saint-Sauveur	J0R	Canada
46	913 East Drive	Lanigan	J7A	Canada
47	6177 Schlimgen Point	Santa Cruz	30560	Mexico
48	32040 Red Cloud Center	Hollywood	33023	United States
49	70991 Red Cloud Way	Shediac	E4P	Canada
50	4 Anderson Avenue	Nashville	37205	United States
51	682 Prentice Parkway	Houston	77293	United States
52	630 Cottonwood Parkway	San Francisco	78490	United States
53	83973 Gina Circle	Lawrenceville	30245	United States
54	61698 Fair Oaks Center	Vista Hermosa	39130	Mexico
55	5 Toban Court	New York City	10060	United States
56	933 Jenifer Parkway	La Concepcion	52105	Mexico
57	52142 Aberg Drive	Providence	02912	United States
58	62325 Butterfield Point	Columbus	43220	United States
59	6 Elm Street	Dallas	75287	United States

aid	street	city	postal_code	country
60	2398 Village Green Drive	Lindavista	70633	Mexico
61	18 Surrey Court	Camlachie	G5Z	Canada
62	20672 Weeping Birch Place	Santa Elena	30807	Mexico
63	3 Ruskin Avenue	Iowa City	52245	United States
64	6824 Esch Alley	La Victoria	95603	Mexico
65	78918 Farwell Alley	Stockton	95205	United States
66	50 Elka Point	Donnacona	G3M	Canada
67	858 Novick Point	Lac La Biche	E1W	Canada
68	92637 Main Plaza	Saint Louis	63180	United States
69	681 Crowley Lane	Orlando	32819	United States
70	0312 Village Drive	Des Moines	50393	United States
71	55349 Green Ridge Parkway	Monroe	71213	United States
72	5006 Lotheville Road	Twyford	LE14	United Kingdom
73	Elm Street	Saguenay	G7K	Canada
74	07 Lighthouse Bay Avenue	El Salitre	58503	Mexico
75	4605 2nd Point	Oklahoma City	73142	United States
76	60281 Clarendon Drive	El Limon	92105	Mexico
77	7 Maple Wood Point	Vista Hermosa	39130	Mexico
78	381 Holmberg Park	Sutton	CT15	United Kingdom
79	07 Golf Court	Houston	77293	United States
80	9 Nancy Plaza	Cuauhtemoc	40068	Mexico
81	23 Gale Drive	Juan N Alvarez	39030	Mexico
82	094 Jenifer Alley	Pasadena	B2J	Canada
83	8 Daystar Road	Tacoma	98405	United States
84	50730 Glacier Hill Avenue	Parrsboro	L2A	Canada
85	30 Warrior Road	20 de Noviembre	93828	Mexico
86	37 Ruskin Circle	Mesa	85205	United States
87	0800 Westend Pass	San Agustin	29480	Mexico
88	17 Mcguire Alley	Kensington	S7K	Canada
89	423 Sycamore Crossing	Jonquière	G7Y	Canada

aid	street	city	postal_code	country
90	98477 Coleman Place	Wilkes Barre	18706	United States
91	248 Becker Alley	Detroit	48232	United States
92	142 Mandrake Center	Denton	M34	United Kingdom
93	71 Cottonwood Terrace	Detroit	48217	United States
94	45281 Sage Junction	Boise	83722	United States
95	04794 Reindahl Parkway	Burgeo	N9A	Canada
96	61404 Ohio Lane	Denton	76210	United States
97	92 Sachtjen Point	Evansville	47737	United States
98	73478 Hoard Point	Washington	20409	United States
99	031 Kenwood Center	La Laguna	51247	Mexico
100	674 Trailsway Court	Port Colborne	L3K	Canada

Table 10: Phone Example Data Table

phid	phone	phone_type	gid	oid
1	6707702050	Home	55	
2	2686847640	Mobile	85	
3	8974034405	Other	84	
4	2146612197	Work		6
5	9971786818	Other	13	
6	2481153325	Mobile		85
7	2073487356	Mobile	16	
8	9289593733	Home	23	
9	4603114742	Other	92	
10	5332914896	Work	11	
11	9573829756	Work		56
12	6983811143	Other		40
13	5587305080	Mobile		42
14	9663460601	Other	81	
15	1469420580	Other		74
16	8463110224	Mobile		4
17	8346294971	Mobile	38	
18	2124979385	Home	71	
19	4724295763	Mobile	36	
20	6138366275	Mobile	59	
21	1049587236	Work	33	
22	4216191521	Home		35
23	4672124625	Home	15	
24	2673538146	Mobile	50	
25	6948470147	Home	75	
26	2422004719	Other	88	
27	1881382322	Mobile	70	
28	3394661197	Work	14	
29	9715823661	Other	32	

phid	phone	phone_type	gid	oid
30	6882792580	Home		65
31	7306044682	Other	71	
32	7732856172	Home	70	
33	4299003047	Mobile		37
34	4249994840	Mobile	41	45
35	2407955416	Work	85	
36	8773051509	Mobile		66
37	7456245618	Mobile		48
38	9971786818	Other	13	
39	2328903472	Other	95	
40	1981422863	Other	1	
41	3739452930	Work	1	
42	9106768111	Other	8	
43	4239539513	Work		1
44	7856568262	Work	53	
45	9193491665	Work	18	
46	9554274795	Other		85
47	9167299711	Work		51
48	1771462251	Other	99	
49	7179464496	Other		4
50	9422281745	Home	43	
51	6934051654	Other	23	
52	5866960308	Other		36
53	3037695904	Mobile		14
54	6162317636	Work	72	
55	9422281745	Home	43	
56	8983553590	Home	70	
57	6804858994	Home	13	
58	1724067554	Home	50	
59	9185386923	Home		18

phid	phone	phone_type	gid	oid
60	6948470147	Home	75	
61	1153680394	Other	97	
62	9289593733	Home	23	
63	1188240110	Work		64
64	8658936979	Mobile	65	
65	2615749244	Home		41
66	1805169564	Other	4	
67	7688747867	Mobile		29
68	4231881146	Other	46	
69	8149205447	Work		54
70	8093015440	Home	27	
71	2422004719	Other	88	
72	5466273238	Home		56
73	2806696188	Work	62	
74	6948635253	Other	27	
75	2938846463	Work		82
76	7597066270	Other		37
77	9603693677	Home		45
78	9653772608	Other	80	
79	4684570922	Home	51	
80	2957802922	Mobile		31
81	6804858994	Home	13	
82	3929915928	Work	80	
83	4341369139	Other	1	
84	4263137593	Other	5	
85	2331938221	Home	8	
86	4616097395	Work		7
87	1533058114	Mobile	10	
88	7298288809	Work		4
89	7725896083	Other	10	

phid	phone	phone_type	gid	oid
90	5555567698	Mobile		12
91	1049587236	Work	33	
92	9351445175	Work	26	
93	3739452930	Work	1	
94	6707702050	Home	55	
95	4206251620	Other		18
96	8586450411	Home	55	
97	9106768111	Other	8	
98	8923838292	Other	27	
99	4131613725	Work	87	
100	3528204391	Work		91
101	8983553590	Home	70	
102	6269589979	Work		54
103	3394661197	Work	14	
104	5113412605	Home		69
105	7194391915	Mobile		18
106	7708323635	Work	26	
107	7694331252	Other	89	
108	7725896083	Other	10	
109	7803952013	Other	19	
110	9464193465	Home	53	
111	1679801461	Home	27	
112	4527886617	Other	22	
113	1153680394	Other	97	
114	8586450411	Home	55	
115	1845679810	Other	8	
116	4233252624	Mobile		98
117	7351562869	Mobile	27	
118	8343861257	Home	51	
119	5603547643	Home		94

phid	phone	phone_type	gid	oid
120	4288186111	Mobile	76	
121	4006681732	Work	38	
122	9863508838	Work	86	
123	5613665400	Work		65
124	9781103482	Work	93	
125	2525774102	Other	56	
126	7803952013	Other	19	
127	1845679810	Other	8	
128	3974920894	Other	86	
129	9207058839	Work	72	
130	4387061242	Work		98
131	5323876067	Other		92
132	8572520795	Home		63
133	8817530646	Other		56
134	8343861257	Home	51	
135	2444406432	Home	84	
136	4582871556	Mobile		67
137	3862879500	Other		2
138	8658936979	Mobile	65	
139	1815997575	Mobile	87	
140	3929915928	Work	80	
141	9072592569	Home	13	
142	3792421124	Other	30	
143	4596099382	Work	78	
144	6475528948	Work		22
145	9193491665	Work	18	
146	9028145666	Work		72
147	2124979385	Home	71	
148	1749711206	Other	46	
149	4527886617	Other	22	

phid	phone	phone_type	gid	oid
150	4231881146	Other	46	
151	4829253932	Mobile		52
152	8256860067	Mobile	44	
153	2138786568	Work	58	
154	5042416181	Work	80	
155	2759263525	Mobile		38
156	2549355111	Home		52
157	2301953695	Work		33
158	5403413026	Work		89
159	7856568262	Work	53	
160	4194595792	Work		4
161	6948635253	Other	27	
162	6564220725	Other		35
163	7712385217	Mobile		39
164	8626641971	Home		5
165	2644438045	Work		69
166	1771462251	Other	99	
167	5042416181	Work	80	
168	6138366275	Mobile	59	
169	2138786568	Work	58	
170	1981422863	Other	1	
171	3209572232	Mobile	10	
172	9072592569	Home	13	
173	5031368578	Work	87	
174	6721126822	Home		60
175	2806696188	Work	62	
176	9663460601	Other	81	
177	8346294971	Mobile	38	
178	2525774102	Other	56	
179	4288186111	Mobile	76	

phid	phone	phone_type	gid	oid
180	1093662090	Work	74	
181	3974920894	Other	86	
182	8093015440	Home	27	
183	8923838292	Other	27	
184	9863508838	Work	86	
185	2073487356	Mobile	16	
186	5436479084	Home		15
187	8628859628	Other		81
188	3792421124	Other	30	
189	9715823661	Other	32	
190	8425943678	Work		2
191	6055118079	Mobile	9	
192	4991523150	Home	80	
193	8809732578	Home		44
194	1093662090	Work	74	
195	2266184428	Mobile		16
196	5443193165	Mobile		22
197	7228862360	Home		50
198	1536507571	Home	8	
199	1724067554	Home	50	
200	2673538146	Mobile	50	
201	6055118079	Mobile	9	
202	9351445175	Work	26	
203	4681847654	Work	24	
204	8158249171	Mobile	25	
205	7351562869	Mobile	27	
206	9704524614	Mobile		71
207	6951280111	Mobile	64	
208	5496719371	Home		68
209	4672124625	Home	15	

phid	phone	phone_type	gid	oid
210	9653772608	Other	80	
211	2639315690	Other		52
212	1533058114	Mobile	10	
213	1805169564	Other	4	
214	9345463638	Mobile		89
215	2699787781	Home		88
216	6162317636	Work	72	
217	4603114742	Other	92	
218	8558696988	Home		89
219	3038499225	Work		22
220	1326063673	Other		99
221	6909179835	Mobile		67
222	9392809428	Work		82
223	9447807241	Other		61
224	8056632210	Work	21	
225	7354455692	Work	57	
226	2328903472	Other	95	
227	2331938221	Home	8	
228	2094823992	Home	19	
229	8684585855	Mobile	26	
230	7354455692	Work	57	
231	6951280111	Mobile	64	
232	1679801461	Home	27	
233	8974034405	Other	84	
234	1536507571	Home	8	
235	7306044682	Other	71	
236	2305571160	Work	98	38
237	1749711206	Other	46	
238	3918701974	Work		55
239	2872804282	Mobile	98	

phid	phone	phone_type	gid	oid
240	5843015708	Other	35	
241	4684570922	Home	51	
242	4131613725	Work	87	
243	9705530744	Other	67	
244	8056632210	Work	21	
245	4991523150	Home	80	
246	3209572232	Mobile	10	
247	9455474468	Other		52
248	5536809817	Home	10	
249	5233548799	Home		49
250	3533009376	Other		41
251	4596099382	Work	78	
252	9781103482	Work	93	
253	2872804282	Mobile	98	
254	5314566768	Work		52
255	3214479328	Home	53	
256	5536809817	Home	10	
257	7672953105	Other		24
258	9993646676	Home	53	
259	2116614889	Other		32
260	5332914896	Work	11	
261	4724295763	Mobile	36	
262	7708323635	Work	26	
263	5843015708	Other	35	
264	4341369139	Other	1	
265	2407955416	Work	85	
266	6176684455	Mobile		55
267	1553059794	Work		74
268	3668901247	Mobile		28
269	5432367460	Work		10

phid	phone	phone_type	gid	oid
270	8032300367	Other		27
271	4006681732	Work	38	
272	8158249171	Mobile	25	
273	5031368578	Work	87	
274	1881382322	Mobile	70	
275	2627406623	Home		27
276	1771544431	Other		88
277	1414888137	Work		76
278	2094823992	Home	19	
279	8684585855	Mobile	26	
280	7694331252	Other	89	
281	6934051654	Other	23	
282	5833234444	Home		95
283	7732856172	Home	70	
284	1458540383	Mobile		91
285	8256860067	Mobile	44	
286	2686847640	Mobile	85	
287	9207058839	Work	72	
288	9705530744	Other	67	
289	9993646676	Home	53	
290	6896540206	Other		36
291	4548516587	Home		33
292	2444406432	Home	84	
293	5961465703	Mobile		33
294	4263137593	Other	5	
295	6089712072	Work		61
296	3214479328	Home	53	
297	9464193465	Home	53	
298	9968295278	Other		59
299	4681847654	Work	24	

phid	phone	phone_type	gid	oid
300	1815997575	Mobile	87	

Table 11: Includes Example Data Table

pid	aid
52	77
36	14
30	95
67	67
18	70
100	5
42	78
42	54
41	98
34	52
98	33
16	52
14	88
96	8
83	22
98	27
5	7
45	74
4	27
83	27
99	77
82	14
31	55
88	96
22	34
20	98
42	60
88	81
22	49

pid	aid
27	47
89	53
2	95
90	91
38	14
24	51
58	30
28	13
86	5
59	39
68	91
56	73
75	16
36	93
16	84
66	32
42	34
75	14
19	22
90	73
55	84

Table 12: Selects Example Data Table

bid	pid
16	56
81	8
34	67
85	53
96	26
5	92
77	22
58	47
50	73
74	32
80	52
13	89
89	90
83	92
61	9
92	89
16	88
78	76
37	51
77	90
33	61
21	16
66	88
7	66
35	5
38	71
47	22
20	69
89	20

bid	pid
62	38
75	30
98	48
6	9
8	67
71	34
82	91
59	86
70	100
40	85
59	56
78	59
81	11
75	53
65	49
66	95
2	56
46	13
68	9
15	33
66	39

Table 13: LivesAt Example Data Table

gid	aid	start_date	end_date
60	30	2018-06-22	2018-08-05
85	80	2018-09-19	
58	53	2018-07-10	
56	44	2018-04-16	
76	42	2018-02-25	
99	59	2018-06-12	
26	62	2017-11-25	
30	35	2018-09-20	
35	17	2018-10-08	
58	11	2017-11-27	2017-12-04
78	81	2018-01-01	2018-06-23
20	31	2018-09-07	
62	42	2018-06-07	
30	15	2017-11-12	
10	76	2018-08-08	
93	68	2018-02-10	2018-06-09
10	55	2018-08-17	
97	63	2017-12-07	
18	74	2018-09-14	2018-10-04
83	67	2018-08-05	
31	26	2018-05-14	
57	20	2018-01-08	
17	56	2018-07-26	
73	70	2017-11-09	2018-06-14
6	26	2018-01-16	
33	99	2018-03-12	
26	58	2018-09-27	2018-07-05
89	79	2017-11-18	
60	50	2018-05-20	

gid	aid	start_date	end_date
78	33	2018-05-14	
83	70	2018-09-17	2018-06-28
21	57	2018-03-19	
77	56	2018-04-24	
48	71	2018-09-14	
91	22	2018-09-23	2017-12-23
5	100	2018-05-15	
8	89	2018-07-11	
53	77	2018-08-04	2018-03-16
32	4	2018-05-21	2018-06-29
10	5	2018-08-17	
66	19	2017-11-29	2018-08-18
79	19	2018-08-18	2017-11-20
63	95	2018-10-08	
50	30	2018-05-19	
21	34	2018-02-19	
89	98	2018-02-16	
57	82	2018-08-20	
73	47	2018-03-04	2018-02-09
6	88	2018-05-06	
5	91	2018-09-09	
74	15	2017-12-05	
80	69	2018-09-27	
47	40	2018-05-30	
57	5	2018-06-11	2018-04-26
18	60	2018-07-22	
40	64	2018-02-16	
65	97	2018-02-04	
55	20	2018-06-17	
14	39	2018-05-11	

gid	aid	start_date	end_date
51	41	2018-06-14	
18	15	2018-08-12	
46	43	2018-05-28	
77	47	2018-10-13	
74	21	2017-12-12	
60	21	2018-06-28	
21	79	2018-03-31	2017-11-15
50	28	2018-05-14	2017-11-04
77	90	2018-03-12	
7	78	2018-02-14	
3	5	2018-03-21	
59	31	2018-07-09	
79	12	2018-01-23	
48	69	2018-09-25	2017-12-07
78	77	2018-06-22	
48	30	2018-01-10	
29	78	2018-07-11	2018-10-18
88	48	2018-03-04	2017-12-12
57	55	2017-11-10	
3	91	2018-04-08	2017-11-23
13	25	2018-05-28	
17	65	2018-03-21	
58	13	2018-03-23	
100	67	2018-09-12	
73	13	2017-11-25	
65	54	2018-02-06	
37	53	2018-02-01	
92	22	2018-06-07	
59	5	2018-06-01	
86	57	2017-12-21	

gid	aid	start_date	end_date
33	4	2018-06-18	2018-07-17
81	88	2018-02-22	
42	77	2018-01-31	
72	20	2018-02-18	
1	57	2017-10-29	
91	29	2018-09-22	
35	97	2018-07-28	
87	8	2018-06-23	
71	41	2018-03-14	2018-03-13
25	77	2018-09-19	
61	38	2018-08-07	

Table 14: OperatesAt Example Data Table

oid	aid	start_date	end_date
42	65	2018-06-11	
23	55	2018-05-01	
7	98	2018-07-17	
64	37	2018-05-12	
25	86	2018-04-03	2018-03-14
98	38	2018-09-11	2018-02-27
53	22	2018-01-27	
57	80	2018-08-10	2018-01-26
72	92	2018-09-04	
28	24	2018-06-01	
80	82	2018-01-15	
55	79	2018-05-01	2017-11-01
65	20	2018-05-16	
29	51	2018-05-30	2018-07-02
92	58	2017-10-27	
32	44	2018-08-19	
3	23	2018-05-16	
12	19	2018-07-27	2017-11-15
22	22	2017-11-24	
45	70	2018-06-08	2018-04-07
53	17	2017-12-27	2018-05-15
46	70	2018-04-06	
20	85	2018-06-22	
60	15	2018-06-27	2018-02-01
25	43	2018-02-10	
27	21	2018-10-05	2017-12-05
28	20	2017-10-31	
50	80	2018-04-29	
33	10	2018-06-03	

oid	aid	start_date	end_date
99	21	2018-01-05	
93	12	2018-01-02	
43	36	2018-08-12	2017-12-04
77	88	2018-03-17	
46	64	2018-08-14	
44	60	2018-04-18	2018-10-21
95	41	2018-05-12	
25	92	2018-04-25	2017-12-29
63	85	2018-09-26	2018-05-20
91	49	2018-03-04	
44	17	2018-04-27	
34	68	2018-09-16	2018-09-19
16	55	2018-07-19	2017-12-27
10	71	2017-11-15	
54	42	2018-09-17	
44	51	2017-10-27	
49	51	2018-02-16	
94	65	2018-07-11	
40	20	2017-12-27	
20	37	2018-04-26	
1	2	2018-06-23	
32	96	2018-04-30	2018-08-20
88	54	2018-07-15	
47	69	2018-04-11	2018-05-17
73	1	2018-07-31	
82	60	2017-12-04	2017-12-23
75	88	2018-09-20	
57	10	2018-04-07	
65	5	2018-06-08	
64	68	2018-08-14	

oid	aid	start_date	end_date
20	52	2018-08-10	
45	57	2018-08-31	
81	72	2018-07-21	2018-10-04
7	23	2018-08-04	2017-11-30
74	28	2018-01-13	
3	94	2018-08-26	
83	40	2018-05-17	
69	94	2018-04-17	2017-12-01
39	72	2017-12-21	2018-06-03
61	63	2017-10-29	2018-10-22
11	32	2018-05-14	2018-08-11
93	39	2018-07-25	
4	37	2018-02-11	2018-01-23
64	31	2018-06-05	
26	37	2018-09-18	
43	90	2017-12-30	
17	24	2018-04-15	
37	28	2018-01-13	
48	90	2018-05-25	
71	58	2017-11-23	2018-07-13
27	97	2018-01-20	2017-11-03
81	62	2018-04-18	
48	94	2017-11-20	
2	82	2018-03-20	
17	65	2017-10-28	
62	26	2018-09-23	
82	69	2018-01-16	
10	86	2018-08-18	
16	38	2018-02-13	
69	89	2018-07-06	

oid	aid	start_date	end_date
92	75	2017-12-09	2018-02-15
1	37	2018-08-30	
25	44	2018-01-17	
28	18	2018-08-25	
44	34	2018-09-30	
27	83	2018-09-22	
79	9	2018-07-01	
7	48	2018-06-03	
50	12	2018-06-20	2018-02-22
23	40	2018-08-17	
73	81	2018-08-12	2018-09-14

2.4 Sample Queries of Database

In the next few sections, we will be doing *sample queries* for Palms Vacation Properties. Firstly, we wil llist some non-trivial and challenging questions. Then, we will transform those questions into *Relational Algebra Expressions*, *Tuple Relational Caculus*, and *Domain Relational Calculus*. Further descriptions are provided in the next pages that follow.

2.4.1 Design of Queries

Now that we have a better understanding of our relational database, we move on to the design of queries. Here, we are converting our relational schema to mathematical expressions. There are three formal methods for querying our design.

2.4.2 Relational Algebra Expressions for Queries

Relational Algebra (RA) is very important for formal definitons for relational model operations. The *RA* is a method that uses set of operations for retrieving tuples from relational database. The *relational expressions algebra* uses mathematical concepts and a procedual method that uses functional operators. Some operators that it includes range from select, join, cross product and others.

1. List all properties with at least 2 bookings in San Francisco with prices more than $1000.

$$\pi\, p\, *\, \Big(\sigma\, property\ x\ booking\ x\ booking\ x\ includes\ x\ includes\ x\ address\ x\ address \Big)$$

$$p \quad b1 \quad b2 \quad i1 \quad i2 \quad a1 \quad a2$$

$$p.pid = b1.pid \wedge b1.pid \wedge b1.pid = i1.pid \wedge i1.aid = a1.aid \wedge$$

$$p.pid = b2.pid \wedge b2.pid \wedge b2.pid = i2.pid \wedge i2.aid = a2.ai \wedge$$

$$a1.city = \text{'San Francisco'} \wedge a1.city = a2.city$$

$$b1.price > 100 \wedge b2.price > 100 \wedge b1.bid \neq b2.bid$$

2. List properties that have exactly one booking cancellation with booking price greater than $1000.

$$property \leftarrow \pi\, p.* \left(\sigma_{p.pid = b1.pid \wedge p.pid = b2.pid \wedge b1.booking\_status = \text{'Canceled'} \wedge b1.daily\_price > 1000 \wedge b2.daily\_price > 1000 \wedge b1.bid \neq b2.bid} \left(property\ p \times booking\ b1 \times booking\ b2 \right) \right)$$

$$\pi\ p.* \left(\sigma_{p.pid = b.pid = b.price > 1000} \left(property\ p \times booking\ b \right) \right) - property$$

3. List guests who have at least one booking on each of the properties which appears in each of every city.

$$allcitieswithproperties \leftarrow \sigma_{p.pid,\, a.city} \left(Property\ p \times Includes\ i \times Address\ a \right)_{p.pid = i.pid \wedge i.aid = a.aid}$$

$$guestcities \leftarrow \sigma_{g.gid,\, p.pid,\, a.city} \left(Guest\ g \times Booking\ b \times Property\ p \times Includes\ i \times Address\ a \right)_{g.gid = b.gid \wedge p.pid = i.pid \wedge i.aid = a.aid}$$

$$\pi_{g.*}\, Guest\ g \times \pi_{agc.gid} \left(guestcities\ agc \div allcitieswithproperties\ acp \right)_{g.gid = agc.gid}$$

4. List all properties with least one booking with more than 3 views and with a price of more than $2000. (Luxury properties)

$$\pi\, p.* \left(\sigma_{p.pid = b.pid \wedge v1.name \,!= v2.name \wedge v2.name \,!= v3.name \wedge v1.name \,!= v3.name \wedge p.price > 2000} \left(property\ p \times booking\ b \times view\ v1 \times view\ v2 \times view\ v3 \right) \right)$$

5. List all owners names that have properties with at least 2 rooms with at least size 300 sqft.

$$\pi_{ow.*}\left(\sigma_{ow.oid = of.oid \wedge p1.pid = of.pid \wedge p2.pid = of.pid \wedge p1.pid \neq p2.pid \wedge p1.size > 300 \wedge p2.size > 300 \wedge p1.rooms > 2 \wedge p2.rooms > 2}\left(\rho_{p1}(property) \times \rho_{p2}(property) \times \rho_{ow}(owner) \times \rho_{of}(offers)\right)\right)$$

6. List guests that have never booked a property in Italy.

$$guestsandaddr \leftarrow \sigma_{g.*,\ a.*}\left(\rho_{g}(Guest) \times \rho_{b}(Booking) \times \rho_{p}(Property) \times \rho_{i}(Includes) \times \rho_{a}(Address)\right)_{g.gid = b.gid \ \wedge\ b.pid = p.pid \ \wedge\ i.pid = p.pid \ \wedge\ i.aid = a.aid}$$

$$\pi_{g.*}\left(\pi_{g.*}(\rho_{a}(guestsandaddr)) - \pi_{g.*}(\sigma_{b.city = 'Italy'}(\rho_{b}(guestsandaddr)))\right)$$

7. List all properties with at least 4-star rating with price > 1000. (Excellent Rated properties).

$$\pi_{p.*}\left(\sigma_{p.pid = pr.pid \wedge pr.rating > 4 \wedge p.price > 1000}\left(\rho_{p}(property) \times \rho_{pr}(property\_rating)\right)\right)$$

8. List guests who have booked in Elm Street, same street during Jane Doe's visit.

$$guestbookaddr \leftarrow \sigma_{g.*,b.*,\ a.*}\left(\rho_{g}(Guest) \times \rho_{b}(Booking) \times \rho_{p}(Property) \times \rho_{i}(Includes) \times \rho_{a}(Address)\right)_{g.gid = b.gid \ \wedge\ b.pid = p.pid \ \wedge\ i.pid = p.pid \ \wedge\ i.aid = a.aid}$$

$$\pi_{g.*}\left(\pi_{*}(\rho_{a}(guestbookaddr)) \times \pi_{*}(\sigma_{b.firstname = 'Jane' \wedge b.lastname = 'Doe' \wedge b.street = "Elm\ Street"}(\rho_{b}(guestbookaddr)))_{a.street = b.street \wedge a.gid \,!= b.gid \wedge a.cin \leq b.cin \wedge a.cout \geq b.cout}\right)$$

9. List guests who have booked in the same property as John Smith's stay between 01/ 01/ 2017 - 01/05/2017.

$$guestinproperty \leftarrow \sigma_{g.*,b.*} \left(Guest^{g} \times Booking^{b} \times Property^{p} \right)$$
$$g.gid = b.gid \wedge b.pid = p.pid$$

$$\pi_{g.*} \left(\pi_{*} \left(guestinproperty \right)^{a} \times \pi_{*} \left(guestinproperty \right)^{b} \right)$$
$$b.firstname = 'John' \wedge$$
$$b.lastname = 'Smith' \wedge$$
$$b.cin \geq 01/01/2017' \wedge$$
$$b.cout \leq '01/05/2017$$
$$a.gid \mathrel{!}= b.gid \wedge$$
$$a.cin \leq b.cin \wedge a.cout \geq b.cout$$

10. List all available properties which have at least an ocean and city view from 03/24/2019 to 04/12/2019. (Popular summer properties)

$$propertiesunavail \leftarrow \sigma_{p.*} \left(Guest^{g} \times Booking^{b} \times Property^{p} \right)$$
$$g.gid = b.gid \wedge b.pid = p.pid \wedge$$
$$b.cin \geq '03/24/2019 \wedge b.cout\ '04/12/2019' \wedge$$
$$b.'booking\_status \mathrel{!}= 'Void' \wedge b.booking\_status \mathrel{!}= 'Canceled'$$

$$\pi_{p.*} \left(\sigma_{p.*} \left(booking^{b} \times property^{p} \times view^{v1} \times view^{v2} \right) - propertiesunavail \right)$$
$$b.pid = p.pid \wedge$$
$$v1.view = 'Ocean' \wedge v2.view = 'City'$$

2.4.3 Tuple Relational Calculus Expressions for Queries

The *Tuple Relational Calculus (TRC)* provides a declarative language specifying relational queries. In *relational calculus* describes the set of tupes that are retrieved and it does not specify which order the operations are to be retrierved. Furthermore, this is why its considered a*nonprocedural* language . Also, *TRC* uses free variables, bound variables, logical expressions, quantifiers and more.

1. List all properties with at least 2 bookings in San Francisco with prices more than

$1000.

$$\{p \mid Property(p) \wedge (\exists b1)(\exists b2)(\exists i1)(\exists i2)(\exists a1)(\exists a2) \wedge ($$
$$Booking(b1) \wedge Includes(i1) \wedge Address(a1) \wedge ($$
$$b1.pid = i1.pid \wedge i1.aid = a1.aid) \wedge ($$
$$Booking(b2) \wedge Includes(i2) \wedge Address(a2) \wedge ($$
$$b2.pid = i2.pid \wedge i2.aid = a2.aid) \wedge ($$
$$b1.bid \neq b2.bid \wedge$$
$$a1.city = 'San\ Francisco \wedge$$
$$a1.city = a2.city \wedge$$
$$b1.daily\_price > 1000 \wedge$$
$$b2.daily\_price > 1000$$
$$)))\}$$

2. List properties that have exactly one booking cancellation with booking price greater than $1000.

$$\{p \mid Property(p) \wedge (\exists b1)(Booking(b) \wedge ($$
$$b1.status = 'Canceled' \wedge b.pid = p.pid$$
$$b1.daily\ price > 1000 \wedge \neg(\exists b2)($$
$$Booking(b2) \wedge ($$
$$b2.price > 1000 \wedge b1.bid \neq p1.bid \wedge$$
$$b1.bid = p.pid$$
$$)))\}$$

3. List guests who have at least one booking on each of the properties which appears in each of every city.

$$\{ g \mid Guest(g) \wedge (\forall a)(\exists b)(\exists p)(\exists i)(\exists a)($$
$$Property(p) \wedge Booking(b) \wedge (b.gid = b.gid\, g \wedge b.pid = p.pid)$$
$$\rightarrow Includes(i) \wedge Address(a) \wedge ($$
$$i.pid = p.pid \wedge i.bid = b.bid$$
$$)) \}$$

4. List all properties with least one booking with more than 3 views and with a price of more than $2000. (Luxury properties)

$$\{p \mid Property(p) \wedge (\exists b)(\exists v1)(\exists v2)(\exists v3) ($$
$$Booking(b1) \wedge View(v1) \wedge View(v2) \wedge View(v3) \wedge ($$
$$p.pid = b.bid \wedge$$
$$v1.name \neq v2.name \wedge$$
$$v2.name \neq v3.name \wedge$$
$$v1.name \neq v3.name \wedge$$
$$b.\, daily\_price > 2000$$
$$)) \}$$

5. List all owners names that have properties with at least 2 rooms with at least size 300 sqft.

$$\{ o.firstname,\ o.lastname \mid Owner(o) \wedge (\exists of)(\exists p)($$
$$Offers(of) \wedge Property(p) \wedge ($$
$$o.oid = of.oid \wedge of.pid = p.pid$$
$$p.rooms > 2 \wedge p.size > 300$$
$$)) \}$$

6. List guests that have never booked a property in Italy.

$$\{ g \mid Guest(g) \wedge (\forall b)\,(Booking(b) \wedge (b.gid = g.gid)$$
$$\rightarrow \neg(\exists p)\,(\exists b2)\,(\exists i)(\exists a)\,($$
$$property(p) \wedge booking(b2) \wedge includes(i) \wedge address(a) \wedge ($$
$$p.pid = b.pid' \wedge i.pid = p.pid \wedge a.aid = i.aid \wedge$$
$$p.country = 'Italy' \wedge b2.bid \neq b.bid$$
$$)\,)\,\}$$

7. List all properties with at least 4-star rating with price $>$ 1000. (Excellent Rated properties).

$$\{p \mid Property(p) \wedge (\exists pr)(PropertyRating(pr) \wedge$$
$$pr.rating > 4 \wedge$$
$$p.price > 1000 \wedge p.pid$$
$$)\}$$

8. List guests who have booked in Elm Street, same street during Jane Doe's visit.

$$\{ g \mid Guest(g1) \wedge (\exists g2)(\exists b1)(\exists b2)(\exists p1)(\exists p2)(\exists i1)(\exists i2)(\exists a1)(\exists a2)($$
$$Guest(g1) \wedge Guest(g2) \wedge Booking(b1) \wedge Booking(b2) \wedge$$
$$Property(p1) \wedge Property(p2) \wedge Includes(i1) \wedge Includes(i2) \wedge$$
$$Address(a1) \wedge Address(a2) \wedge ($$
$$g1.gid = b1.gid \wedge b1.aid = p1.pid \wedge i1.pid = p1.pid \wedge$$
$$i1.aid = a1.aid \wedge$$
$$g2.gid = b2.gid \wedge b2.aid = p2.pid \wedge i2.pid = p2.pid \wedge$$
$$i2.aid = a2.aid\,) \wedge ($$
$$g2.firstname = 'Jane' \wedge g2.lastname\ 'Doe' \wedge$$
$$g1.gid\ != g2.gid$$
$$a1.street = a2.street \wedge a2.street = 'Elm\ Street' \wedge$$
$$b2.cin <= b1.cin \wedge b2.cout >= b1.cout$$
$$)))\,\}$$

9. List guests who have booked in the same property as John Smith's stay between 01/ 01/ 2017 - 01/05/2017.

$$\{ g \mid Guest(g1) \wedge (\exists g2)(\exists b1)(\exists b2)(\exists p1)(\exists p2)(Guest(g1) \wedge Guest(g2) \wedge Booking(b1) \wedge Booking(b2) \wedge Property(p1) \wedge Property(p1) \wedge (g1.gid = b1.gid \wedge b1.aid = p1.pid \wedge g2.gid = b2.gid \wedge b2.aid = p2.pid) \wedge (g2.firstname = 'John' \wedge g2.lastname 'Smith' \wedge g1.gid != g2.gid \wedge p1.pid = p2.pid \wedge b2.cin <= '01/01/2017' \wedge b2.cout >= '01/05/2017')) \}$$

10. List all available properties which have at least an ocean and city view from 03/24/2019 to 04/12/2019. (Popular summer properties)

$$\{ p \mid Property(p1) (\exists v1)(\exists v2) (View(v1) \wedge View(v2)((v1.name = 'Ocean' \wedge v2.name = 'City') \wedge \neg(\exists b)(Booking(b) \wedge (b.cin >= '03/24/2019' \wedge b.cout <= '03/24/2019)' \wedge (b.booking\_status \neq 'Cancelled \wedge b.booking\_status \neq 'Void'))) \}$$

2.4.4 Domain Relational Calculus Expressions for Queries

Domain relational Calculus (DRC) is in the *type of variables* used in formulas. In this method, the variables range over single vaues from domain attributes rather than having variables range over tuples.

Like TRC is it made up from atomic, but instead of of the atoms being tuples, they are the individual values within the list of value of each tuple.

1. List all properties with at least 2 bookings in San Francisco with prices more than $1000.

$$\{< p, d, r, s, pr > | Property(p, d, r, s, pr) \wedge Include(p, a) \wedge$$
$$Address(a, \_, 'San\ Francisco', \_, \_) \wedge$$
$$Booking(b1, \_, \_, \_, \_, \_, \_, p, \_) \wedge$$
$$Booking(b2, \_, \_, \_, \_, \_, \_, p, \_) \wedge ($$
$$b1.bid\ !=\ b2.bid$$
$$pr\ >\ 1000$$
$$) \}$$

2. List properties that have exactly one booking cancellation with booking price greater than $1000.

$$\{< p, d, r, s, pr > | Property(p, d, r, s, pr) \wedge (\forall b1)($$
$$Booking(b1, > 1000, \_, \_, \_, 'Canceled', \_, p, \_) \wedge$$
$$\neg(\exists b2)(Booking(b2, > 1000 \_, \_, \_, \_, 'Canceled' \_, \_, p, \_) \wedge ($$
$$b1.bid\ \neq\ b2.bid$$
$$)) \}$$

3. List guests who have at least one booking on each of the properties which appears in each of every city.

$$\{\ < g, f, m, l, e > \ | Guest(g, f, m, l, e) \wedge (\forall a)(\exists b)(\exists p)($$
$$Property(p, \_, \_, \_, \_\ ) \wedge Booking(b, \_, \_, \_, \_, \_, \_, g, p)$$
$$\rightarrow Includes(p, a1) \wedge Address(a1, \_, \_, \_\ )$$
$$) \}$$

4. List all properties with least one booking with more than 3 views and with a price of more than $2000. (Luxury properties)

$$\{< p,\ d,\ r,\ s,\ pr > \mid Property(p,\ d,\ r,\ s,\ pr)\ \wedge (\exists b)($$
$$Booking(b, > 2000,\ \_,\ \_,\ \_, \_\ ,\ \_,\ p,\ \_)\ \wedge$$
$$(\exists v1)(\exists v2)(\exists v3)(View(p,\ v1) \wedge (View(p,\ v2) \wedge (View(p,\ v3)\ \wedge ($$
$$v1 \neq v2\ \wedge v2\ \neq v3\ \wedge v1\ \neq v3$$
$$)\ )\ )\ \}$$

5. List all owners names that have properties with at least 2 rooms with at least size 300 sqft.

$$\{< o,f,m,l,e > \mid Owner(o,f,m,l,e)\ \wedge\ (\exists p)\ ($$
$$Offers(o,\ p,\ \_,\ \_)\ \wedge$$
$$Property(p, \_,\ \_,\ > 2, > 300,\ \_\ )$$
$$)\ \}$$

6. List guests that have never booked a property in Italy.

$$\{\ < g,f,\ m,\ l,\ e\} \mid Guest(g,\ f,\ m,\ l,\ e)\ \wedge \neg(\exists b)\ ($$
$$Booking(b, \_,\ \_,\ \_,\ \_,\ \_,\ p,\ g)\ \wedge$$
$$Property(p,\ \_,\ \_,\ \_,\ \_) \wedge$$
$$Includes(p,\ a)\ \wedge Address(a, \_,\ \_,\ \_,\ 'Italy')$$
$$)\ \}$$

7. List all properties with at least 4-star rating with price $>$ 1000. (Excellent Rated properties).

$$\{< p,\ d,\ r,\ s,\ price > \mid Property(p,\ d,\ r,\ s,\ price)\ \wedge$$
$$(\exists pr)\ (PropertyRating(pr, > 4, \_, \_, p, \_, \_)\ \wedge$$
$$price\ >\ 1000$$
$$)\ \}$$

8. List guests who have booked in Elm Street, same street during Jane Doe's visit.

$$\{ < g,f,m,l,e > \mid Guest(g,f,m,l,e) \wedge ($$
$$(\exists b1)(\exists p1)(\exists a1)($$
$$Booking(b1,\_,\_,ci1,co1,\_,\_,g1,p1) \wedge Property(p1,\_,\_,\_,\_) \wedge$$
$$Includes\,(p1,a1) \wedge Address(a1,s1,\ \_,\ \_,\ \_) \wedge ($$
$$(\exists b2)(\exists p2)(\exists a2)(\ Guest(\neq g,\ 'Jane',\ \_,\ 'Doe',\ \_) \wedge ($$
$$Booking(b2,\_,\_,ci2,co2,\_,\_,g,p) \wedge Property(p2,\_,\_,\_,\_) \wedge$$
$$Includes\,(p2,a2) \wedge Address(a,s2,\ \_,\ \_,\ \_) \wedge ($$
$$s1\ =\ s2 \wedge s2 =\ 'Elm\ Street')$$
$$)))))\}$$

9. List guests who have booked in the same property as John Smith's stay between 01/ 01/ 2017 - 01/05/2017.

$$\{ < g,f,m,l,e > \mid Guest(g,f,m,l,e) \wedge ($$
$$(\exists b1)(\exists p1)($$
$$Booking(b1,\_,\_,'01/01/2017','01/05/2017',\_,\_,g1,p1) \wedge$$
$$Property(p1,\_,\_,\_,\_) \wedge ($$
$$(\exists b2)(\exists p2)(\ Guest(\neq g,\ 'John',\ \_,\ 'Smith',\ \_) \wedge ($$
$$Booking(b2,\_,\_,'01/01/2017','01/05/2017'\_,\_,g2,p2) \wedge$$
$$Property(p2,\_,\_,\_,\_)$$
$$)))\}$$

10. List all available properties which have at least an ocean and city view from 03/24/2019 to 04/12/2019. (Popular summer properties)

$$\{ <p, d, r, s, pr> \mid Property(p, d, r, s, pr) \wedge (View(p, 'Ocean') \wedge View(p, 'City') \wedge (\neg(\exists b)(Booking(b, \_, \_, \leq '03/24/2019', \geq '04/12/2019', stat, \_, p, \_) \wedge (stat \neq 'Void' \wedge stat \neq 'Canceled')))))\}$$

3 Phase 3: Postgres SQL Database Management System

In this third phase we create a logical and physical database with Postgres DBMS. However, before we can convert our relational database to a physical databse we must explain the process.

For this reason, we analyze topics such as the normalization of relations, explain the purpose and functionality of PostgreSQL, define schema objects in PostgreSQL, decode PostgreSQL relational schema, showcase the transfer of queries from Phase 2 in PostgreSQL, and briefly discuss a data loader method we used to load large amounts of data into our implementation. After we illustrate and resolve the aforementioned, we will successfully convert our relational database to PostgreSQL.

3.1 Normalization of Relations

In this section, we discuss a method called *normalization* of relations. The concept of database *normalization* helps us reorganize our Palms Vacation Properties relational database. We reorganize it to fit under the various normal forms.

Furthermore, we must restructure our relational shema in order to reduce data redundancy and refine data integrity. It is important to note that before we can transform our logical database to a physical database, we must improve our database schema. Hence, we must analyze the quality of our design by using normalization methods. In addition to this, we further discuss some anomalies and problems that may arise if our relations are not normalized in the next few pages.

3.1.1 Normalization and Normal Forms

In this subsection, we specifically, focus on what a normalization is and what normal forms are. We define the concepts of nomalization and normalize using normal forms. Not only that but we also discuss anomalies and the types of anomalies that affect our database.

Description of Normalization and Normal Forms

Normalization was first proposed by Codd (1972), and this normalization process takes a relation schema through a series of test cases in order to *approve* if it qualifies and meets specific normal form. The normalization of data is considered a process of analyzing a given relation schema based on their foreign keys and primary keys to achieve acceptable properties of *minimizing* redudancy, insertion, deletion, and update anomalies. In a similar manner, we

can consider *normalization* to be a *filtering* process to make our Palms Vacation Properties database a more structured and quality design.

On the other hand, if any of our relation schemas do not meet acceptable conditions, then they are split into smaller realation schemas that meet the test and possess acceptable properties. But that will occur only if the conditions in our relational schema does not pass the *normal form* test cases.

A normal form is of a relation that refers to the highest form of condition that a relation meets, and this indicates the degree to which it has been normalized. Our goal is to simplify, filter, and process our relation schemas and remove any redundancies by testing the nomalization of each of the relation schemas. These are the normal forms discussed further:

- First normal form (1NF)
- Second normal form (2NF)
- Third normal form (3NF)
- Boyce-Codd normal form (BCNF)

First Normal Form

First Normal Form is also known as *1NF* for short and part of the formal definition of a relation in the relational model. The first normal form has the domain of an attribute that must include only atomic values. Also, that same value of any attribute in a tuple has to be a single value from the domain of the attribute. This same normal form does not allow nested tuples of values in the relation schemas.

In order for the first normal form to be true, all the attributes inside a relation must be single and not contain any multiple values and they must be atomic. However, there are ways to decompose a relation schema that is not first normal form. For example, the multi-value attribute can be a separate relation that can contain the primary key as a foreign key attribute. The method mentioned is to be considered the best one mosty because it does not suffer from redundancy as much as the other two methods. Also, there is no limit placed on how many values it can have.

Second Normal Form

Second normal form is also known as *2NF* for short and based on *full functional depency*. The definition of this is form is that if a relation schema R is in second normal form and if every nonprime attribute A in R is fully functional depent on the primary key of R. To simplify this definition, we can narrow it down to if a set of attributes Y is functionally dependent on another set of values of X and if the set of values for X can be mapped to a set of Y values then the values of X can determine the values of Y.

To test for second normal form we need to test for functional dependecies whose attributes are part of the primary key. Then, if the primary key contains a single attribute then there is no need to apply the test. At any case, if the relational schema fails the test then it can be eventually normalized but first it must be broken into smaller relation schemas and have its primary keys be the subsets of the primary original key.

Third Normal Form

Third normal form is also known as *3NF* for short and its basis is on *transitive depency*. The definition of the third normal form is that a relation schema R is in 3NF if it satifies second normal form and no nonprime attribute of R is transitively dependent on the primary key. If the test fails the third normal form, then we can further normalize the relational schema by decomposing it into two 3NF relation schemas. Moreover, in that case we can have any functional dependency where the left-hand side is part of the primary key but we can decompose the relations into new relations to solve any functional dependency issues that may arise.

Boyce-Codd Normal Form

Boyce-Codd Normal Form is also kwown as *BCNF* for short and it is supposed to be a simpler for of the Third Normal Form, however it was found to be stricter. Therefore, every realtion in the Boyce-Codd Normal Form is also found in 3NF. The difference lies in that a relation in 3NF may not necessarily be found in BCNF.

The definition of BCNF tells us that a relation schema R is in BCNF if whenever a nontrivial functional depency of X is in A, that holds in R, then X is a superkey of R. All previous normal forms must be satisfied before the relation schema can be satifiesd for BCNF. If the relation schema fails the BCNF test, then the schema can be further decomposed into relations where the nonprime attributes on the left side become prime attributes for new relation schemas.

Anomalies of Poor Normalization

Our goal for a relational schema design is to avoid anomalies and redundancies. But in some cases, anomalies are inevitable because the integrity of the data we have stored may eventually degrade. Hence, we have anomalies that occur and the results are not represented without redundancies.

Moreover, anomalies are caused when we have too much redundancy in our database. Redundancies are caused for various reasons. For example, we can have a table that can suffer from poor construction, such as the dependency of property and address. These are the types of anomalies that occur and that we discuss in detail:

- Insertion Anomalies
- Update Anomalies

- Deletion Anomalies
- Modification Anomalies

Insertion Anomalies

Insertion Anomalies occurs when we have to insert data into a database and it is not possible since other data is already in its place. For example, in our database we require that in order to be give a *guest review*, a *guest* must first be added and must have a confirmed *booking* in order to be able to leave a *guest review*. In simple terms, this means that a there can not be a *guest rating* without having a *guest*, then we have an *insert* anomaly.

We can split the anomalies in two separate types. Firstly, if we insert a new tuple representing a relation that is not joined with another relation then the attribute values for the joined relation must be the same for both tuples in order to prevent redundancy. Secondly, if we have to insert a tuple that represents a relation that is not joined in any of the other relations then the attribute values for the other relation schema has to be set to NULL. However, we still have issues because there is data that is incorrent and data that is either missing or not known.

Update Anomalies

Update Anomalies occur when we store natural joins of base relations. This type of anomalies happen when we have to *update* information that is already stored in the database. For example, if we have to update information about an *owner* due to change of ownership. If we store the information in the same table, the information will be incorrect and we will have multiple owners that may actually not own the property or have the previous owners co-owning the property. This causes a problem because the user may not know who the owners are and the information retrieved may be incorrect.

Furthermore, since the information updated can become easily inconsistent, it is imperative that if a set of tupes are resepresented in a relation that is joined with other relations, then the attributes of that relations must properly and correctly appear in all of the tuples. But if any of the attribute values are changed in one tuple, then we must also change them in allof the tuples affected by this *update* in order for our data to remain consistent.

Deletion Anomalies

Deletion Anomalies is quite similar to *insertion anomaly* in that it affects all tuples related to the attributes values. If we have a set of tuples that represents a relation that is joined to few other relations and we *delete* the tuples then we have corrupted any record of that relation and it is removed from the database.

In doing this, we even run into the risk of deleting undesirable information and this causes other wanted information to be deleted as well. For example, if we delete information about

a specific *owner*, we run the risk of deleting everything that is related to that owner and other properties that may be related to the owner.

Modification Anomalies
Modification Anomalies occur if we change the value of one of the attributes. If we have to update the tuples, then we must update all of the values that are affected, otherwise the data becomes inconsistent. If we fail to update the information correctly then there will be two different values in the tuples. Modification anomalies are named as such because it entails inserting, deleting and updating of data in our database table.

3.1.2 Normal Forms for Palms Vacation Properties

Now that we have a clear understanding of how to use *normalization*, we can reorganize our relational database with their respective normal forms. In the next few pages, we analyze and decode each relation schema to check if it meets the requirements to fall under *Third Normal Form(3NF)*. Also, we further examine the relation schema by joining two relation schemas and check if there are any anomalies that occur in the process.

Owner

- **Functional Dependencies:**

  - FD1. **oid** → { firstname, middlename, lastname,..., email }

- **Candidate Keys:**

  oid

- **Normal Form Tests:**

  - 1NF- Satisfied because attributes are atomic.
  - 2NF- Satisfied because primary key has a single attribute.
  - 3NF- Satisfied because non-prime attributes do not depend on other non-prime attributes.
  - BCNF- Satisfied because left side of any functional dependencies is a candidate key.

  BCNF is satisfied, hence no anomalies.

Guest

- **Functional Dependencies:**

- FD1. **gid** → { firstname, middlename, lastname,..., email }

- **Candidate Keys:**

  gid

- **Normal Form Tests:**

  - 1NF- Satisfied because attributes are atomic.
  - 2NF- Satisfied because primary key has a single attribute.
  - 3NF- Satisfied because non-prime attributes do not depend on other non-prime attributes.
  - BCNF- Satisfied because left side of any functional dependencies is a candidate key.

  BCNF is satisfied, hence no anomalies.

Property

- **Functional Dependencies:**

  - FD1. **pid** → { description, rooms,..., price }

- **Candidate Keys:**

  pid

- **Normal Form Tests:**

  - 1NF- Satisfied because attributes are atomic.
  - 2NF- Satisfied because primary key has a single attribute.
  - 3NF- Satisfied because non-prime attributes do not depend on other non-prime attributes.
  - BCNF- Satisfied because left side of any functional dependencies is a candidate key.

  BCNF is satisfied, hence no anomalies.

Amenity

- **Functional Dependencies:**

  None

- **Candidate Keys:**

  pid

- **Normal Form Tests:**

  - 1NF- Satisfied because attributes are atomic.
  - 2NF- Satisfied because primary key has a single attribute.
  - 3NF- Satisfied because non-prime attributes do not depend on other non-prime attributes.
  - BCNF- Satisfied because left side of any functional dependencies is a candidate key.

  BCNF is satisfied, hence no anomalies.

Attraction

- **Functional Dependencies:**

  None

- **Candidate Keys:**

  pid

- **Normal Form Tests:**

  - 1NF- Satisfied because attributes are atomic.
  - 2NF- Satisfied because primary key has a single attribute.
  - 3NF- Satisfied because non-prime attributes do not depend on other non-prime attributes.
  - BCNF- Satisfied because left side of any functional dependencies is a candidate key.

  BCNF is satisfied, hence no anomalies.

View

- **Functional Dependencies:**

  None

- **Candidate Keys:**

  pid

- **Normal Form Tests:**

  - 1NF- Satisfied because attributes are atomic.
  - 2NF- Satisfied because primary key has a single attribute.
  - 3NF- Satisfied because non-prime attributes do not depend on other non-prime attributes.
  - BCNF- Satisfied because left side of any functional dependencies is a candidate key.

  BCNF is satisfied, hence no anomalies.

Booking

- **Functional Dependencies:**

  - FD1. **bid** $\rightarrow$ { price }

- **Candidate Keys:**

  bid, pid, gid, confirm_nr

- **Normal Form Tests:**

  - 1NF- Satisfied because attributes are atomic.
  - 2NF- Satisfied because primary key has a single attribute.
  - 3NF- Satisfied because non-prime attributes do not depend on other non-prime attributes.
  - BCNF- Satisfied because left side of any functional dependencies is a candidate key.

  BCNF is satisfied, hence no anomalies.

Property_Rating

- **Functional Dependencies:**

  None

- **Candidate Keys:**

  prid

- **Normal Form Tests:**

  - 1NF- Satisfied because attributes are atomic.
  - 2NF- Satisfied because primary key has a single attribute.
  - 3NF- Satisfied because non-prime attributes do not depend on other non-prime attributes.
  - BCNF- Satisfied because left side of any functional dependencies is a candidate key.

  BCNF is satisfied, hence no anomalies.

Address

- **Functional Dependencies:**

  - FD1. **aid** → {street, ..., country}

- **Candidate Keys:**

  aid

- **Normal Form Tests:**

  - 1NF- Satisfied because attributes are atomic.
  - 2NF- Satisfied because primary key has a single attribute.
  - 3NF- Satisfied because non-prime attributes do not depend on other non-prime attributes.
  - BCNF- Satisfied because left side of any functional dependencies is a candidate key.

  BCNF is satisfied, hence no anomalies.

Phone

- **Functional Dependencies:**

  - FD1. **phid** → {phone_number, phone_type, gid, oid}

- **Candidate Keys:**

  phid, gid, oid

- **Normal Form Tests:**

  - 1NF- Satisfied because attributes are atomic.
  - 2NF- Satisfied because primary key has a single attribute.
  - 3NF- Satisfied because non-prime attributes do not depend on other non-prime attributes.
  - BCNF- Satisfied because left side of any functional dependencies is a candidate key.

  BCNF is satisfied, hence no anomalies.

Offers

- **Functional Dependencies:**

  - FD1. **oid, pid** $\rightarrow$ {start_date, end_date }

- **Candidate Keys:**

  oid, pid

- **Normal Form Tests:**

  - 1NF- Satisfied because attributes are atomic.
  - 2NF- Satisfied because primary key has a single attribute.
  - 3NF- Satisfied because non-prime attributes do not depend on other non-prime attributes.
  - BCNF- Satisfied because left side of any functional dependencies is a candidate key.

  BCNF is satisfied, hence no anomalies.

Includes

- **Functional Dependencies:**

  None

- **Candidate Keys:**

  pid, aid

- **Normal Form Tests:**

  - 1NF- Satisfied because attributes are atomic.
  - 2NF- Satisfied because primary key has a single attribute.
  - 3NF- Satisfied because non-prime attributes do not depend on other non-prime attributes.
  - BCNF- Satisfied because left side of any functional dependencies is a candidate key.

  BCNF is satisfied, hence no anomalies.

Selects

- **Functional Dependencies:**

  None

- **Candidate Keys:**

  bid, pid

- **Normal Form Tests:**

  - 1NF- Satisfied because attributes are atomic.
  - 2NF- Satisfied because primary key has a single attribute.
  - 3NF- Satisfied because non-prime attributes do not depend on other non-prime attributes.
  - BCNF- Satisfied because left side of any functional dependencies is a candidate key.

  BCNF is satisfied, hence no anomalies.

LivesAt

- **Functional Dependencies:**

  - FD1. **gid,aid** $\rightarrow$ {start_date, end_date }

- **Candidate Keys:**

  gid, aid

- **Normal Form Tests:**

  - 1NF- Satisfied because attributes are atomic.
  - 2NF- Satisfied because primary key has a single attribute.
  - 3NF- Satisfied because non-prime attributes do not depend on other non-prime attributes.
  - BCNF- Satisfied because left side of any functional dependencies is a candidate key.

  BCNF is satisfied, hence no anomalies.

Example of Poorly Normalized Relation: Booking_Property
Booking and *Property* is a relation schema we create by natural join to use as an example of *Booking_ Property*. By creating this example, we can further explain what its functional dependencies are, candidate keys, natural form and why this combination may present possible anomalies.

- **Relation:**

  Booking_Property(**bid**, daily_price, booking_date, cin, cout, booking_status, confirm_nr, gid, pid, description, rooms, size, price)

- **Functional Dependencies:**

  - FD1. **bid** → { daily_price, booking_date, cin, cout, booking_status, confirm_nr }

  - FD2. **pid** → { description, rooms, size, price, gid (foreign key), pid (foreign key) }

- **Candidate Keys:**

  bid, pid, confirm_nr

- **Normal Form Tests:**

  Under the normal form tests, the relation is not satisfied under 3NF, but it is satisfied under 1NF and 2NF. The reasoning behind this is that there are functionalities that depends on nonprime attributes and the transitively depends on the primary key.

- **Possible anomalies**:

  After analyzing and processing the normal form tests, we can see that there is the INSERTION anomaly because if we insert new Properties, there is no way to do so unless we insert NULLs for *gid*, which is a primary key. Then, we also find UPDATE anomaly in that if we change booking, we have to update it and change it everywhere else. Lastly, we have the DELETE anomaly, if we delete reservations for a room in a property, we loose the room information. This tells us that this is a very poor normalized example overall.

3.2 PostgreSQL: Purpose and Functionality

Previously, we considered and explained normalization, types of normal forms, and types of anomalies. Now that we have normalized our design we can dive into creating and implementing a physical database.

We begin the implementation process by using PostgreSQL also known as simply Postgres. Postgres is an open source object-relational database management system. We will use Postgres to further reveal and illustrate our physical database.

PostgreSQL functions allows us to carry out multiple operations that would usually take several queries in a single function within our database. Also, PostgreSQL has a user interface that the we can interact with, create, update and query data. PostgreSQL includes built-in support for indexes, schemas, data types, user-defined objects, inheritance, and others such as referential constraints that include foreign key contraints and others.

3.3 Schema Objects in PostgreSQL Database

In this section we further break down the implementation process of our physical database and the schema objects in PostgreSQL database. Also, we list some schema objects created in our database. We commence by examining the syntax statements and provide examples found in Palms Vacation Properties database for the following:

- Tables
- Functions
- Triggers
- Sequence Generators
- Indexes

Tables

Tables in PostgreSQL cosists of rows and colums. It is much like the tables that we have worked with. The number and order of the columns of the tables are fixed and each column is named accordingly such as *rooms*, *city*, *firstname*. The number of rows is variable and reflects the data that is stored.

Each column of the table has its own data type and within the data type we have set constraints for the possible values that it can be assigned (such as TIMESTAMP and VARCHAR). PSQL has its own set-built in data types but database designers can create and define their own data types. For example, we can use the built-in *integer* for whole numbers, *numeric* for fractional numbers, *text* for character strings and so on.

Syntax:

```
create table <tablename> (
  <column-def-1>,
  ...
  <column-def-N>,
  <table constraints>
) <column-def> := <column-name><column-datatype><column-constraints>

<table constraints> :=
  constraints <constraint-name> primary key (<column-name>),
  foreign key (<column-name>) references <table-name> (<column-name>),
  unique (<column-name>),
  check <boolean-expression>
```

Examples in this implementation:

- Address
- Phone
- Owner
- Property
- Booking

Views

Views are the results of any query stored as virtual tables. This defines a view of a query. These do not necessarily store data but simply generates the representation of data. In this

manner, a database designer can have control over what data is available, its presentation and how it should be formatted. Also, views are dynamically recreated whenever there is a modification information. CREATE OR REPLACE VIEW is quite similar since if a view of the same name exists in the database then it is replaced. The newly created query generates the same columns of the previous view query but may have additionally columns respective to the output columns.

Syntax:

```
create (or replace) view <view_name> AS <select statement/query>
```

Function
Functions are stored as blocks of PSQL code that can be run by using scripts. They give more reusability and are able to perform repetitive manipulations on data. However, in order to define a new function, a user must have user previledge on the language and argument types.

Syntax:

```
create (or replace) function <function_name> begin <statements> end
```

Triggers
Triggers are call back functions that are invoked when specific events occur in the database.

Syntax:

```
create (or replace) trigger <trigger_name> after <event_name> on <table_name>
begin <statements> end
```

Sequence Generators
Sequence Generators are created through mathematical functions to showcase a sets of unique values. The sequence generators sometimes are used to create unique primary key attributes. Also, these same sequence generators helps us ensure that the unique primary key values are used for the new information such as tuples being inserted even if there are many users requesting at the same time.

Syntax:

```
create sequence <sequence name>
  minvalue <minimum value>
  maxvalue <maximum value>
```

```
start with <starting number>
increment by <increment amount>
cache <cache size>
```

Examples in this implementation:

- Address.aid
- Phone.phid
- Owner.oid
- Property.pid
- Booking.bid
- Guest.gid
- PropertyRating.prid

Indexes

Indexes are special look up tables that facilitiate and increase the speed of data retrieval. In simpler terms, an index is similar to a pointer that points to data in a table. Indexes are quite common when it comes to enhancing the database performance. This alows the server to find and retrieve specific rows in a speedier manner than without using an index. However, indexes add overhead and should be used sparingly.

3.4 Postgres Relational Schema and Contents

Relation Owner - DESCRIBE

```
palms=> \d owner
                                  Table "public.owner"
   Column   |         Type          | Collation | Nullable |              Default
------------+-----------------------+-----------+----------+-----------------------------------
 oid        | integer               |           | not null | nextval('owner_oid_seq'::regclass)
 firstname  | character varying(32) |           |          |
 middlename | character varying(32) |           |          |
 lastname   | character varying(32) |           |          |
 email      | character varying(64) |           |          |
Indexes:
    "owner_pk" PRIMARY KEY, btree (oid)
Referenced by:
    TABLE "offers" CONSTRAINT "offers_oid_fk" FOREIGN KEY (oid) REFERENCES owner(oid)
    TABLE "operatesat" CONSTRAINT "operatesat_oid_fk" FOREIGN KEY (oid) REFERENCES owner(oid)
    TABLE "phone" CONSTRAINT "phone_oid_fk" FOREIGN KEY (oid) REFERENCES owner(oid)
```

SELECT * FROM Owner:

```
 oid |  firstname  | middlename |  lastname   |              email
-----+-------------+------------+-------------+---------------------------------
   1 | Dacia       | Jules      | Stidson     | jstidson0@yale.edu
   2 | Nikolos     |            | Whittier    | nwhittier1@multiply.com
   3 | Harlan      | Noellyn    | Lieb        | nlieb2@hostgator.com
   4 | Elisa       | Garland    | Scrowton    | gscrowton3@gmpg.org
   5 | Gilburt     | Cherye     | Kennsley    | ckennsley4@cam.ac.uk
   6 | Spike       |            | Swithenby   | tswithenby5@buzzfeed.com
   7 | Jean        | Cherilynn  | Jeandel     | cjeandel6@si.edu
   8 | Fiorenze    | Erek       | Greve       | egreve7@google.es
   9 | Corie       | Celina     | Hansod      | chansod8@mashable.com
  10 | Brenden     | Lief       | Kosiada     | lkosiada9@berkeley.edu
  11 | Weylin      | Sorcha     | Keppe       | skeppea@gravatar.com
  12 | Brena       | Moritz     | Newgrosh    | mnewgroshb@topsy.com
  13 | Manya       | Coriss     | Gonoude     | cgonoudec@gizmodo.com
  14 | Tessi       |            | Gullivent   | jgulliventd@unicef.org
  15 | Culley      | Quillan    | Gout        | qgoute@adobe.com
  16 | Aeriel      |            | Denmead     | rdenmeadf@a8.net
  17 | Vale        |            | Gadaud      | kgadaudg@godaddy.com
  18 | Jonis       | Hieronymus | Carillo     | hcarilloh@usnews.com
  19 | Flory       | Roley      | Brightey    | rbrighteyi@cargocollective.com
  20 | Anestassia  | Isadora    | Agett       | iagettj@tamu.edu
  21 | Melita      | Margit     | McFeate     | mmcfeatek@mapquest.com
  22 | Raeann      | Aindrea    | Hartridge   | ahartridgel@acquirethisname.com
  23 | Derril      |            | Van der Son | avandersonm@tinypic.com
  24 | Bernice     |            | Fleckness   | cflecknessn@google.com.br
  25 | Maddy       | Irene      | Dufore      | iduforeo@cnn.com
  26 | Alyda       | Becki      | Musgrave    | bmusgravep@goo.ne.jp
  27 | Talia       | Axe        | Bathow      | abathowq@microsoft.com
  28 | Talia       | Andriette  | Fernely     | afernelyr@nbcnews.com
  29 | Kara-lynn   | Joaquin    | Penton      | jpentons@pen.io
  30 | Othello     | Trix       | De Carlo    | tdecarlot@youtube.com
  31 | Orelle      |            | Muttock     | mmuttocku@51.la
  32 | Louella     |            | Sieghart    | asieghartv@fda.gov
  33 | Jerrilyn    | Simone     | Salthouse   | ssalthousew@cargocollective.com
  34 | Deerdre     | Tedmund    | Mc Combe    | tmccombex@webeden.co.uk
  35 | Patrica     | Aeriell    | Mallabar    | amallabary@phpbb.com
  36 | Dre         |            | Karoly      | okarolyz@mapquest.com
  37 | Karlie      |            | Ivamy       | rivamy10@gmpg.org
  38 | Marabel     |            | Bellson     | tbellson11@istockphoto.com
  39 | Tudor       |            | Tasch       | rtasch12@sfgate.com
  40 | Vivianna    | Lurlene    | Droghan     | ldroghan13@ed.gov
  41 | Talia       | Baily      | Flattman    | bflattman14@msn.com
  42 | Aarika      | Brien      | Eloy        | beloy15@discuz.net
  43 | Effie       | Noble      | Shorie      | nshorie16@gov.uk
  44 | Ignaz       | Brannon    | McSharry    | bmcsharry17@cnbc.com
  45 | Tonya       | Iggie      | Domanski    | idomanski18@stanford.edu
  46 | Osbourn     |            | Allmann     | eallmann19@privacy.gov.au
  47 | Sigvard     | Milena     | Scriver     | mscriver1a@columbia.edu
  48 | Valle       | Fidela     | Fairbridge  | ffairbridge1b@surveymonkey.com
  49 | Pierette    | Davidde    | Culshaw     | dculshaw1c@example.com
  50 | Chandler    | Lishe      | Fewless     | lfewless1d@1688.com
  51 | Kellen      | Filippa    | Tribble     | ftribble1e@yale.edu
  52 | Shay        | Chev       | Deville     | cdeville1f@jugem.jp
  53 | Ginevra     |            | Ehlerding   | eehlerding1g@exblog.jp
  54 | Darelle     | Conroy     | Stanney     | cstanney1h@woothemes.com
  55 | Ban         | Allison    | Harradine   | aharradine1i@ycombinator.com
  56 | Artemis     | Lian       | Comello     | lcomello1j@about.me
  57 | Emmalyn     | Clarissa   | Hurche      | churche1k@domainmarket.com
  58 | Thia        | Shaina     | Massingberd | smassingberd1l@webmd.com
  59 | Luce        | Fancy      | Cicullo     | fcicullo1m@vkontakte.ru
  60 | Kiley       | Sophia     | Stennard    | sstennard1n@jiathis.com
  61 | Jerrie      | Elfie      | Newall      | enewall1o@godaddy.com
  62 | Donaugh     | Thornton   | Geleman     | tgeleman1p@moonfruit.com
  63 | Berget      | Faunie     | Comello     | fcomello1q@behance.net
  64 | Antonella   | Rowney     | Tootell     | rtootell1r@jugem.jp
  65 | Rock        | Thorin     | Gavagan     | tgavagan1s@networksolutions.com
  66 | Jillian     | Bartel     | Piddocke    | bpiddocke1t@ucla.edu
```

```
  67 | Barb        | Ricky      | Bussell     | rbussell1u@timesonline.co.uk
  68 | Carole      |            | Blint       | pblint1v@wikispaces.com
  69 | Ericka      | Conan      | Raatz       | craatz1w@seesaa.net
  70 | Alysia      | Legra      | Jeayes      | ljeayes1x@newyorker.com
  71 | Jodie       | Auroora    | Testo       | atesto1y@wikia.com
  72 | Alexander   | Domingo    | Freeman     | dfreeman1z@chicagotribune.com
  73 | Essie       |            | Scarasbrick | mscarasbrick20@apache.org
  74 | Jasen       |            | Frear       | gfrear21@people.com.cn
  75 | Kippy       |            | Mostin      | lmostin22@wufoo.com
  76 | Taite       | Prissie    | Beckensall  | pbeckensall23@angelfire.com
  77 | Maryjane    | Rafael     | Blannin     | rblannin24@deliciousdays.com
  78 | Rori        | Riki       | Vaskov      | rvaskov25@biblegateway.com
  79 | Gifford     |            | Ricoald     | tricoald26@howstuffworks.com
  80 | Anna-maria  |            | Bowler      | jbowler27@dion.ne.jp
  81 | Christiane  |            | Thirst      | athirst28@hostgator.com
  82 | Ermentrude  | Jud        | Schankel    | jschankel29@chron.com
  83 | Albert      | Michael    | Sparshott   | msparshott2a@spiegel.de
  84 | Jedidiah    | Marice     | Whilde      | mwhilde2b@prlog.org
  85 | Kelly       | Townsend   | Spincks     | tspincks2c@archive.org
  86 | Tiffany     | Tarah      | Lines       | tlines2d@ask.com
  87 | Bartholomew |            | Duthy       | dduthy2e@de.vu
  88 | Debi        | Candi      | Gilburt     | cgilburt2f@mayoclinic.com
  89 | Walsh       | Ron        | Farrah      | rfarrah2g@discuz.net
  90 | Brittney    | Jessi      | Treweke     | jtreweke2h@etsy.com
  91 | Nathanael   | Deane      | Bage        | dbage2i@squidoo.com
  92 | Hulda       |            | Paynter     | dpaynter2j@yellowbook.com
  93 | Philbert    | Addia      | McKimmie    | amckimmie2k@homestead.com
  94 | Cordie      | Dagny      | Pounds      | dpounds2l@latimes.com
  95 | Bride       |            | Mergue      | dmergue2m@webnode.com
  96 | Adriane     | Barbi      | Mazey       | bmazey2n@nih.gov
  97 | Galvin      | Jorrie     | Rens        | jrens2o@t.co
  98 | Ag          | Mariel     | Teare       | mteare2p@wiley.com
  99 | Perkin      | Jerry      | Turbat      | jturbat2q@businesswire.com
 100 | Salome      | Archie     | Kinset      | akinset2r@cnbc.com
(100 rows)
```

Relation Guest - DESCRIBE

```
palms=> \d guest;
                                  Table "public.guest"
   Column   |         Type          | Collation | Nullable |              Default
------------+-----------------------+-----------+----------+-----------------------------------
 gid        | integer               |           | not null | nextval('guest_gid_seq'::regclass)
 firstname  | character varying(32) |           |          |
 middlename | character varying(32) |           |          |
 lastname   | character varying(32) |           |          |
 email      | character varying(64) |           |          |
Indexes:
    "guest_pk" PRIMARY KEY, btree (gid)
Referenced by:
    TABLE "booking" CONSTRAINT "booking_gid_fk" FOREIGN KEY (gid) REFERENCES guest(gid)
    TABLE "livesat" CONSTRAINT "livesat_gid_fk" FOREIGN KEY (gid) REFERENCES guest(gid)
    TABLE "phone" CONSTRAINT "phone_gid_fk" FOREIGN KEY (gid) REFERENCES guest(gid)
    TABLE "property_rating" CONSTRAINT "property_rating_gid_fk" FOREIGN KEY (gid) REFERENCES guest(gid)
```

SELECT * FROM Guest:

```
 gid | firstname  | middlename  |   lastname   |                email
-----+------------+-------------+--------------+-----------------------------------
   1 | Laurent    | Viv         | Kingsbury    | vkingsbury0@youtu.be
   2 | Rosa       | Boniface    | Punton       | bpunton1@accuweather.com
   3 | Berti      | Kelsy       | Kybert       | kkybert2@amazon.co.jp
   4 | Jan        |             | Hing         | bhing3@mysql.com
   5 | Joana      | Cordi       | Talks        | ctalks4@about.me
   6 | Valdemar   | Ana         | Fears        | afears5@e-recht24.de
```

```
 7 | Newton     | Donni       | Franklen     | dfranklen6@elegantthemes.com
 8 | Mufinella  |             | Grey         | agrey7@slashdot.org
 9 | Shawn      | Wilone      | Pape         | wpape8@hp.com
10 | Butch      | Ginni       | Meagh        | gmeagh9@addtoany.com
11 | Tiphanie   | Alvan       | Gerssam      | agerssama@dailymotion.com
12 | Say        | Jack        | Pavkovic     | jpavkovicb@newsvine.com
13 | Turner     | Alejandrina | Sarl         | asarlc@oaic.gov.au
14 | Stavro     | Linc        | Farren       | lfarrend@t.co
15 | Jorgan     | Rosabelle   | Lode         | rlodee@mozilla.com
16 | Brooks     |             | Blunsum      | gblunsumf@fc2.com
17 | Jesus      | Bordie      | Sidnell      | bsidnellg@newyorker.com
18 | Isobel     | Antoine     | Bodicum      | abodicumh@skyrock.com
19 | Yancy      | Bell        | Allflatt     | ballflatti@dot.gov
20 | Janel      | Zsa zsa     | Kirkman      | zkirkmanj@china.com.cn
21 | Janet      | Jon         | Birtwhistle  | jbirtwhistlek@quantcast.com
22 | Vince      |             | Wray         | swrayl@ocn.ne.jp
23 | Augusto    | Parnell     | John         | pjohnm@sciencedaily.com
24 | Daniela    |             | Dowdeswell   | ldowdeswelln@jimdo.com
25 | Cyril      | Griffy      | Yard         | gyardo@technorati.com
26 | Candide    | Griselda    | Battelle     | gbattellep@t-online.de
27 | Roch       | Lorry       | Amner        | lamnerq@naver.com
28 | Kerry      | Reider      | Carswell     | rcarswellr@sogou.com
29 | Wallache   | Merell      | Allett       | malletts@issuu.com
30 | Amby       | Miltie      | Lambden      | mlambdent@google.com
31 | Hyacinth   | Fenelia     | Breedy       | fbreedyu@chicagotribune.com
32 | Haleigh    | Aldrich     | Claffey      | aclaffeyv@vimeo.com
33 | Bone       |             | Keary        | dkearyw@gizmodo.com
34 | Jacinda    | Hastie      | Fitzharris   | hfitzharrisx@liveinternet.ru
35 | Ervin      | Ginger      | Muzzullo     | gmuzzulloy@vkontakte.ru
36 | Leland     | Bridie      | Toth         | btothz@constantcontact.com
37 | Cara       | Aluin       | Petruk       | apetruk10@fc2.com
38 | Catherina  |             | Korpolak     | akorpolak11@dell.com
39 | Loree      |             | Crudginton   | kcrudginton12@last.fm
40 | Tripp      |             | Sellars      | rsellars13@webs.com
41 | Frans      | Mattheus    | Swatridge    | mswatridge14@ox.ac.uk
42 | Roxie      | Ulrike      | Joron        | ujoron15@nyu.edu
43 | Eddi       | Sheri       | Mounch       | smounch16@google.ru
44 | Wandis     | Anjanette   | Frise        | afrise17@sogou.com
45 | Darby      |             | Noden        | znoden18@marriott.com
46 | Leopold    | Mitchell    | Meikle       | mmeikle19@technorati.com
47 | Eldridge   | Athene      | Gillman      | agillman1a@tinyurl.com
48 | Miguel     | Eudora      | Dearell      | edearell1b@friendfeed.com
49 | Travers    | Mozes       | Fauning      | mfauning1c@networkadvertising.org
50 | Adella     | Thornton    | Redmond      | tredmond1d@shareasale.com
51 | Sheri      | Garret      | Gyorffy      | ggyorffy1e@china.com.cn
52 | Uta        |             | Norkutt      | dnorkutt1f@phpbb.com
53 | Patty      | Ayn         | Aronstein    | aaronstein1g@blogger.com
54 | Lancelot   |             | Eyton        | deyton1h@geocities.jp
55 | Clarisse   |             | Habbert      | rhabbert1i@umich.edu
56 | Vittoria   | Rodolfo     | Ajsik        | rajsik1j@virginia.edu
57 | Marylin    | Smith       | Tracy        | stracy1k@google.it
58 | Kliment    | Koo         | Gasquoine    | kgasquoine1l@yellowbook.com
59 | Lars       | Teriann     | Ivett        | tivett1m@pcworld.com
60 | Ogdon      | Corette     | Maultby      | cmaultby1n@hud.gov
61 | Clarissa   |             | Urridge      | kurridge1o@google.de
62 | Randie     | Edita       | McCoveney    | emccoveney1p@webnode.com
63 | Wallace    | Ferd        | Sproat       | fsproat1q@flavors.me
64 | Tanhya     |             | McCullum     | bmccullum1r@indiegogo.com
65 | Jerrilee   | Carmencita  | Ketley       | cketley1s@free.fr
66 | Immanuel   | Pooh        | Baskeyfied   | pbaskeyfied1t@slashdot.org
67 | Hermon     |             | Medlin       | emedlin1u@livejournal.com
68 | Kettie     | Brnaba      | Mulvany      | bmulvany1v@google.com.hk
69 | Bette      |             | Canape       | ecanape1w@gnu.org
70 | Brook      | Gaston      | Crusham      | gcrusham1x@diigo.com
71 | Clair      | Tallie      | Dell Casa    | tdellcasa1y@slate.com
72 | Valle      | Therine     | Jarret       | tjarret1z@statcounter.com
73 | Brynn      |             | Penella      | lpenella20@webs.com
74 | Sammie     |             | Mauser       | rmauser21@ucsd.edu
```

```
  75 | Ken        | Julianna    | Crosscombe   | jcrosscombe22@printfriendly.com
  76 | John       |             | Smith        | j.smith.coolio@arizona.edu
  77 | Fayina     | Allyson     | Minchin      | aminchin24@purevolume.com
  78 | Rustin     | Dacy        | McAndie      | dmcandie25@skype.com
  79 | Tanney     |             | Rosbrough    | rrosbrough26@paypal.com
  80 | Dacie      |             | Binder       | abinder27@merriam-webster.com
  81 | Thea       | Hamid       | Bruni        | hbruni28@microsoft.com
  82 | Gypsy      | Giusto      | Gibard       | ggibard29@mayoclinic.com
  83 | Caryn      | Allistir    | Bannon       | abannon2a@meetup.com
  84 | Sallie     |             | Lemerie      | alemerie2b@goodreads.com
  85 | Nichol     | Loydie      | Arniz        | larniz2c@hatena.ne.jp
  86 | Felipe     |             | Gonnel       | kgonnel2d@uiuc.edu
  87 | Emile      | Erica       | Dayes        | edayes2e@dell.com
  88 | Oralee     | Eldridge    | Scade        | escade2f@home.pl
  89 | Karee      | Barry       | O Mulderrig  | bomulderrig2g@example.com
  90 | Kaycee     |             | Purkess      | hpurkess2h@i2i.jp
  91 | Wallas     | Quentin     | Brou         | qbrou2i@chron.com
  92 | Bronson    | Pierrette   | Bein         | pbein2j@blinklist.com
  93 | Phylis     |             | Snuggs       | asnuggs2k@bloomberg.com
  94 | Murvyn     | Amara       | Buckingham   | abuckingham2l@geocities.jp
  95 | Gardener   | Ashby       | Kiff         | akiff2m@hubpages.com
  96 | Cosette    | Yoshiko     | Piotrowski   | ypiotrowski2n@imgur.com
  97 | Vance      |             | Whitlaw      | kwhitlaw2o@joomla.org
  98 | Ferdinande | Karleen     | Seligson     | kseligson2p@twitter.com
  99 | Ynes       | Myrtice     | Osbourne     | mosbourne2q@engadget.com
 100 | Ricca      |             | Ilden        | milden2r@go.com
(100 rows)
```

Relation Property - DESCRIBE

```
palms=> \d property;
                                  Table "public.property"
   Column    |          Type           | Collation | Nullable |                Default
-------------+-------------------------+-----------+----------+---------------------------------------
 pid         | integer                 |           | not null | nextval('property_pid_seq'::regclass)
 description | character varying(5120) |           | not null |
 rooms       | integer                 |           |          |
 size        | integer                 |           |          |
 price       | numeric(15,2)           |           |          |
Indexes:
    "property_pkey" PRIMARY KEY, btree (pid)
Check constraints:
    "property_rooms_check" CHECK (rooms >= 0)
    "property_size_check" CHECK (size >= 0)
Referenced by:
    TABLE "booking" CONSTRAINT "booking_pid_fk" FOREIGN KEY (pid) REFERENCES property(pid)
    TABLE "includes" CONSTRAINT "includes_pid_fk" FOREIGN KEY (pid) REFERENCES property(pid)
    TABLE "offers" CONSTRAINT "offers_pid_fk" FOREIGN KEY (pid) REFERENCES property(pid)
    TABLE "attraction" CONSTRAINT "pid_pk_fk" FOREIGN KEY (pid) REFERENCES property(pid)
    TABLE "amenity" CONSTRAINT "pid_pk_fk" FOREIGN KEY (pid) REFERENCES property(pid)
    TABLE "property_rating" CONSTRAINT "property_rating_pid_fk" FOREIGN KEY (pid) REFERENCES property(pid)
    TABLE "selects" CONSTRAINT "selects_pid_fk" FOREIGN KEY (pid) REFERENCES property(pid)
    TABLE "view" CONSTRAINT "view_pid_fk" FOREIGN KEY (pid) REFERENCES property(pid)
```

SELECT * FROM Property:

```
pid |         description          | rooms | size |  price
-----+------------------------------+-------+------+---------
   1 | turpis enim                  |     2 | 1132 |  236.18
   2 | molestie lorem quisque       |     2 | 2556 |  267.41
   3 | ante ipsum                   |     4 |  963 |  244.59
   4 | ultrices aliquet maecenas    |     1 | 1773 |  324.21
   5 | sociis natoque               |     4 | 2388 |  164.35
   6 | cursus id                    |     3 |  658 |  167.84
   7 | eget tincidunt eget          |       | 2312 |   93.07
```

```
 8 | eget eros elementum          |     2 | 2744 |  156.30
 9 | congue etiam                 |    10 | 2230 |  260.61
10 | volutpat convallis morbi     |       | 2086 |  348.61
11 | interdum venenatis           |     8 | 1252 |  264.63
12 | nisl nunc nisl               |     9 | 1182 |  227.72
13 | eleifend quam                |     5 |  803 |   78.22
14 | amet turpis elementum        |    10 |  794 | 1096.77
15 | dapibus dolor vel            |     2 | 1533 |  167.76
16 | morbi quis tortor            |       | 2462 |  184.92
17 | tristique tortor eu          |     5 |  955 |  340.61
18 | quis orci                    |     9 |  568 |  192.47
19 | in est                       |     5 | 1800 |  244.87
20 | est donec                    |     4 | 2284 |  335.63
21 | odio consequat varius        |     5 | 2988 |  292.94
22 | commodo placerat praesent    |     7 |  977 |  305.79
23 | non velit nec                |    10 |      |  114.72
24 | vestibulum vestibulum        |     1 |  367 |  214.74
25 | interdum venenatis           |     9 | 2545 |  246.44
26 | suscipit a feugiat           |       |      |  150.18
27 | lectus in                    |     5 | 2344 |  258.70
28 | ligula pellentesque ultrices |     4 | 1633 |  224.28
29 | urna ut tellus               |     9 | 1972 |  135.94
30 | id mauris                    |       | 2551 |  291.31
31 | vestibulum sit               |     7 | 1974 |  296.09
32 | mollis molestie lorem        |     7 | 2422 |  164.67
33 | elit sodales scelerisque     |     5 |  817 |  283.58
34 | tristique est et             |     7 | 2725 |   66.09
35 | quis odio consequat          |    10 |  258 |  261.64
36 | justo lacinia                |     3 | 2032 |  166.77
37 | sit amet turpis              |     9 | 2556 |  169.24
38 | posuere cubilia curae        |     3 | 2054 |  275.89
39 | lorem ipsum dolor            |    10 | 2869 |  285.90
40 | imperdiet et commodo         |     6 | 1833 |  319.16
41 | justo sit                    |     5 |  231 |  281.63
42 | phasellus in                 |     3 | 2615 |  123.72
43 | nibh quisque                 |       | 2073 |  138.61
44 | elit proin interdum          |     7 | 1887 |  305.44
45 | primis in                    |    10 |  542 |  266.84
46 | lacinia nisi                 |     6 | 2622 |  249.08
47 | proin eu                     |    10 |  984 |   44.93
48 | sapien quis libero           |     5 | 2428 |  131.21
49 | purus phasellus in           |     9 | 2887 |   65.88
50 | nam congue risus             |       |  913 |   86.90
51 | elit sodales                 |     1 |      |  176.54
52 | ipsum dolor sit              |     2 |      |   77.91
53 | neque vestibulum             |     7 |  965 |   69.86
54 | donec pharetra magna         |     6 | 1046 |  290.70
55 | mattis nibh                  |    10 | 1763 |  239.67
56 | pulvinar nulla pede          |     7 |      |  294.14
57 | sapien ut                    |     7 |  890 |  174.48
58 | pede venenatis               |     1 | 1022 |  139.58
59 | nunc vestibulum ante         |       |  790 |  219.04
60 | amet eleifend pede           |     1 |      |  207.29
61 | lobortis est                 |       |  557 |   41.65
62 | elementum nullam varius      |     4 | 2835 |  233.24
63 | sapien dignissim vestibulum  |     3 | 2110 | 1181.86
64 | morbi odio odio              |       |  438 |  173.01
65 | mattis odio                  |     9 | 2845 |  116.14
66 | aliquet ultrices erat        |     9 | 2587 |  119.71
67 | luctus cum                   |     2 |  297 |  126.65
68 | orci luctus et               |     3 |  249 |  314.00
69 | varius integer ac            |     8 |  865 |  198.34
70 | in porttitor pede            |     2 |  557 |  122.04
71 | elementum nullam varius      |     9 |  811 |  261.95
72 | integer non velit            |     8 |  584 |  294.71
73 | donec ut                     |     8 |      |  277.26
74 | sapien iaculis congue        |       | 2656 |  246.69
75 | diam vitae quam              |     3 | 2764 |  242.96
```

```
  76 | sodales scelerisque mauris   |     8 | 2010 | 1315.80
  77 | odio donec vitae             |       |  500 |  139.11
  78 | integer pede                 |     6 | 2293 |   87.44
  79 | at turpis a                  |     7 | 1703 |  103.35
  80 | aliquet at feugiat           |     9 |  667 |  203.10
  81 | et ultrices                  |     6 | 2936 |  263.14
  82 | interdum mauris              |     4 |  551 |  146.41
  83 | erat quisque                 |     5 |  854 |   81.27
  84 | ac est                       |       | 1398 |   95.03
  85 | consequat lectus in          |       | 1231 |  277.25
  86 | tortor quis                  |     4 | 1687 |  269.52
  87 | mi sit amet                  |    10 | 2725 |  214.32
  88 | luctus et ultrices           |     2 | 1808 |  261.16
  89 | orci luctus                  |    10 |      |  263.81
  90 | scelerisque quam turpis      |       | 1530 |  204.30
  91 | morbi vel lectus             |    10 |      |  346.93
  92 | rutrum at lorem              |    10 |      |  148.22
  93 | rutrum neque                 |     4 |  647 |  331.89
  94 | quis tortor id               |     2 | 2219 |  173.02
  95 | lobortis convallis tortor    |     9 |  312 |  333.46
  96 | consequat dui nec            |     1 | 1759 |  206.85
  97 | augue a                      |     7 | 1154 |  201.17
  98 | sit amet nulla               |     9 |      |  127.83
  99 | vulputate nonummy            |     6 | 2951 |  191.79
 100 | nibh ligula nec              |     1 |  793 |  270.62
(100 rows)
```

Relation Amenity - DESCRIBE

```
palms=> \d amenity
                    Table "public.amenity"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 pid    | integer               |           |          |
 name   | character varying(32) |           | not null |
Indexes:
    "amenity_pid_name_unique" UNIQUE CONSTRAINT, btree (pid, name)
Foreign-key constraints:
    "pid_pk_fk" FOREIGN KEY (pid) REFERENCES property(pid)
```

SELECT * FROM Amenity:

```
 pid |     name
-----+---------------
  25 | Washer
   1 | Roll-in Bed
   4 | Swimming Pool
   9 | Roll-in Bed
  18 | Dryer
   8 | Swimming Pool
  20 | Roll-in Bed
   8 | Roll-in Bed
  13 | Roll-in Bed
   8 | Dryer
  24 | Dryer
   4 | Dryer
  23 | Washer
   1 | Swimming Pool
  21 | Washer
  21 | Roll-in Bed
   5 | Swimming Pool
  14 | Swimming Pool
  20 | Washer
  11 | Roll-in Bed
  16 | Washer
```

```
  15 | Microwave
   7 | Washer
  19 | Roll-in Bed
   4 | Washer
  22 | Roll-in Bed
  13 | Dryer
  10 | Dryer
  22 | Swimming Pool
  20 | Swimming Pool
  13 | Washer
  24 | Washer
   6 | Dryer
  15 | Roll-in Bed
   1 | Dryer
  10 | Washer
   8 | Washer
  15 | Washer
  22 | Washer
  10 | Microwave
  18 | Swimming Pool
   4 | Roll-in Bed
  10 | Roll-in Bed
   9 | Microwave
   6 | Washer
   3 | Microwave
  22 | Microwave
  22 | Dryer
  19 | Dryer
  20 | Microwave
(50 rows)
```

Relation Attraction - DESCRIBE

```
palms=> \d attraction
                    Table "public.attraction"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 pid    | integer               |           |          |
 name   | character varying(32) |           | not null |
Indexes:
    "attractoin_pid_name_unique" UNIQUE CONSTRAINT, btree (pid, name)
Foreign-key constraints:
    "pid_pk_fk" FOREIGN KEY (pid) REFERENCES property(pid)
```

SELECT * FROM Attraction:

```
 pid |    name
-----+-------------
  17 | City Center
  15 | Monuments
   4 | Theme-parks
  20 | Library
  13 | Theme-parks
  19 | Monuments
  18 | Monuments
  23 | Monuments
   8 | Museums
  25 | Museums
   2 | Monuments
   3 | Theme-parks
   4 | City Center
   8 | Monuments
  16 | City Center
  24 | City Center
   1 | Monuments
```

```
  10 | Museums
  24 | Theme-parks
  24 | Museums
  14 | Theme-parks
  15 | Museums
   7 | City Center
  24 | Library
  22 | Library
  18 | Theme-parks
  24 | Monuments
   4 | Museums
  21 | City Center
  11 | Library
   4 | Monuments
  10 | Library
  11 | City Center
   5 | Theme-parks
  22 | Museums
  19 | Theme-parks
  12 | Monuments
  15 | City Center
  21 | Museums
   6 | Library
   2 | Museums
  10 | Theme-parks
   7 | Theme-parks
  21 | Monuments
  19 | City Center
   9 | Library
   3 | City Center
   6 | Museums
  14 | Museums
  16 | Library
(50 rows)
```

Relation View - DESCRIBE

```
palms=> \d view
                    Table "public.view"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 pid    | integer               |           |          |
 name   | character varying(32) |           | not null |
Indexes:
    "view_pid_name_unqiue" UNIQUE CONSTRAINT, btree (pid, name)
Foreign-key constraints:
    "view_pid_fk" FOREIGN KEY (pid) REFERENCES property(pid)
```

SELECT * FROM View:

```
 pid |   name
-----+----------
   8 | Beach
  13 | Mountain
  25 | Desert
   6 | Vineyard
  47 | Beach
  13 | Vineyard
  47 | City
   9 | Beach
  14 | Lake
  14 | Nature
  19 | Nature
  21 | City
   4 | Desert
```

```
  11 | Ocean
  21 | Vineyard
  55 | City
  25 | Lake
  17 | Lake
  20 | Mountain
  24 | Nature
   9 | Vineyard
  10 | Vineyard
  15 | Nature
  22 | Vineyard
  11 | Mountain
  67 | City
   3 | Vineyard
  70 | City
   2 | City
   8 | Vineyard
  17 | Desert
  17 | Mountain
   2 | Ocean
  15 | City
  25 | Ocean
  70 | Vineyard
  12 | City
  13 | Lake
  25 | Mountain
  11 | City
  15 | Ocean
   5 | Desert
  12 | Beach
  25 | Vineyard
  96 | City
  13 | Nature
  10 | Desert
  97 | City
  21 | Mountain
  18 | Mountain
(50 rows)
```

Relation Booking - DESCRIBE

```
palms=> \d booking
                                        Table "public.booking"
     Column     |            Type             | Collation | Nullable |               Default
----------------+-----------------------------+-----------+----------+-------------------------------------
 bid            | integer                     |           | not null | nextval('booking_bid_seq'::regclass)
 daily_price    | numeric                     |           |          |
 booking_date   | timestamp without time zone |           | not null |
 cin            | timestamp without time zone |           | not null |
 cout           | timestamp without time zone |           | not null |
 booking_status | booking_status              |           | not null |
 confirm_nr     | character varying(64)       |           |          |
 pid            | integer                     |           |          |
 gid            | integer                     |           |          |
Indexes:
    "booking_pk" PRIMARY KEY, btree (bid)
    "booking_confirm_nr_unqiue" UNIQUE CONSTRAINT, btree (confirm_nr)
Check constraints:
    "booking_check" CHECK (cout > cin)
    "booking_daily_price_check" CHECK (daily_price >= 0::numeric)
Foreign-key constraints:
    "booking_gid_fk" FOREIGN KEY (gid) REFERENCES guest(gid)
    "booking_pid_fk" FOREIGN KEY (pid) REFERENCES property(pid)
Referenced by:
    TABLE "property_rating" CONSTRAINT "property_rating_bid_fk" FOREIGN KEY (bid) REFERENCES booking(bid)
    TABLE "selects" CONSTRAINT "selects_bid_fk" FOREIGN KEY (bid) REFERENCES booking(bid)
```

SELECT * FROM Booking:

```
 bid | daily_price |    booking_date     |         cin         |        cout         | booking_status |
 ↪  confirm_nr | pid | gid
-----+-------------+---------------------+---------------------+---------------------+----------------+------------+-----+-----
   1 |      286.12 | 2018-05-07 00:00:00 | 2018-10-24 15:00:00 | 2018-10-28 12:00:00 | Cancelled      |
   ↪  |  59 |  64
   2 |      139.19 | 2018-10-21 00:00:00 | 2018-10-24 15:00:00 | 2018-10-29 12:00:00 | Pending        |
   ↪  |  66 |  29
   3 |      174.13 | 2018-04-12 00:00:00 | 2018-11-06 15:00:00 | 2018-11-08 12:00:00 | Void           |
   ↪  |  21 |  52
   4 |      200.82 | 2018-08-07 00:00:00 | 2018-09-29 15:00:00 | 2018-10-04 12:00:00 | Pending        |
   ↪  |  41 |  35
   5 |      153.47 | 2018-06-29 00:00:00 | 2018-09-28 15:00:00 | 2018-09-30 12:00:00 | Completed      |
   ↪  59e819b2   |  76 |  52
   6 |       60.64 | 2018-07-01 00:00:00 | 2018-10-19 15:00:00 | 2018-10-25 12:00:00 | Void           |
   ↪  |  57 |  73
   7 |       129.1 | 2018-06-15 00:00:00 | 2018-10-28 15:00:00 | 2018-11-06 12:00:00 | Completed      |
   ↪  930d2ea6   |  94 |  43
   8 |      225.39 | 2018-09-17 00:00:00 | 2018-10-02 15:00:00 | 2018-10-11 12:00:00 | Void           |
   ↪  |  75 |  98
   9 |      103.98 | 2018-03-02 00:00:00 | 2018-09-22 15:00:00 | 2018-09-30 12:00:00 | Reserved       |
   ↪  |  64 |  37
  10 |      263.19 | 2018-08-08 00:00:00 | 2018-09-21 15:00:00 | 2018-09-27 12:00:00 | Completed      |
  ↪  bcbaffec   |  40 |  46
  11 |      286.76 | 2018-05-26 00:00:00 | 2018-10-09 15:00:00 | 2018-10-14 12:00:00 | Pending        |
  ↪  |   3 |  57
  12 |      335.89 | 2018-06-30 00:00:00 | 2018-11-11 15:00:00 | 2018-11-13 12:00:00 | Reserved       |
  ↪  | 100 |  76
  13 |      328.78 | 2018-05-13 00:00:00 | 2018-10-06 15:00:00 | 2018-10-13 12:00:00 | Void           |
  ↪  |  31 |  50
  14 |       188.9 | 2018-06-03 00:00:00 | 2018-10-01 15:00:00 | 2018-10-07 12:00:00 | Cancelled      |
  ↪  |  52 |  99
  15 |      149.92 | 2018-04-21 00:00:00 | 2018-10-11 15:00:00 | 2018-10-13 12:00:00 | Completed      |
  ↪  b771aa80   |  89 |  45
  16 |      226.13 | 2018-04-18 00:00:00 | 2018-10-20 15:00:00 | 2018-10-30 12:00:00 | Reserved       |
  ↪  |  71 |  76
  17 |      304.88 | 2018-07-30 00:00:00 | 2018-09-19 15:00:00 | 2018-09-20 12:00:00 | Reserved       |
  ↪  |  41 |  67
  18 |      316.97 | 2018-04-02 00:00:00 | 2018-10-12 15:00:00 | 2018-10-15 12:00:00 | Void           |
  ↪  |  47 |  89
  19 |      259.79 | 2018-04-21 00:00:00 | 2018-10-12 15:00:00 | 2018-10-14 12:00:00 | Void           |
  ↪  |  76 |   1
  20 |      1222.5 | 2018-05-24 00:00:00 | 2018-10-13 15:00:00 | 2018-10-18 12:00:00 | Pending        |
  ↪  |  54 |  86
  21 |       82.97 | 2018-06-12 00:00:00 | 2018-09-12 15:00:00 | 2018-09-15 12:00:00 | Reserved       |
  ↪  |  34 |  53
  22 |       215.6 | 2018-03-29 00:00:00 | 2018-09-30 15:00:00 | 2018-10-06 12:00:00 | Completed      |
  ↪  fdf70f9b   |  22 |  81
  23 |      114.05 | 2018-05-06 00:00:00 | 2018-11-01 15:00:00 | 2018-11-10 12:00:00 | Reserved       |
  ↪  |  18 |  62
  24 |       49.91 | 2018-04-08 00:00:00 | 2018-10-12 15:00:00 | 2018-10-13 12:00:00 | Completed      |
  ↪  f7e96a27   |  80 |  57
  25 |       97.66 | 2018-06-27 00:00:00 | 2018-10-08 15:00:00 | 2018-10-11 12:00:00 | Reserved       |
  ↪  |  96 |  31
  26 |      164.79 | 2018-06-17 00:00:00 | 2018-09-28 15:00:00 | 2018-10-07 12:00:00 | Completed      |
  ↪  fe4d3ed5   |  20 |  91
  27 |      328.16 | 2018-02-13 00:00:00 | 2018-09-26 15:00:00 | 2018-10-01 12:00:00 | Void           |
  ↪  |  87 |  46
  28 |      336.46 | 2018-02-10 00:00:00 | 2018-10-04 15:00:00 | 2018-10-05 12:00:00 | Completed      |
  ↪  97ed550b   |  57 |  88
  29 |      290.12 | 2018-03-26 00:00:00 | 2018-10-27 15:00:00 | 2018-10-28 12:00:00 | Completed      |
  ↪  87836d26   |  68 |  87
  30 |      272.02 | 2018-07-22 00:00:00 | 2018-11-09 15:00:00 | 2018-11-13 12:00:00 | Pending        |
  ↪  |  19 |  47
  31 |      137.36 | 2018-04-27 00:00:00 | 2018-10-06 15:00:00 | 2018-10-09 12:00:00 | Cancelled      |
  ↪  |  33 |  45
  32 |      279.75 | 2018-03-16 00:00:00 | 2018-10-03 15:00:00 | 2018-10-05 12:00:00 | Reserved       |
  ↪  |  68 |  38
```

```
33 |      283.67 | 2018-05-22 00:00:00 | 2018-10-26 15:00:00 | 2018-11-04 12:00:00 | Reserved       |
↪  |  12 |   1
34 |      163.53 | 2018-05-18 00:00:00 | 2018-09-17 15:00:00 | 2018-09-21 12:00:00 | Pending        |
↪  |  96 |   9
35 |     1328.18 | 2018-01-08 00:00:00 | 2018-11-04 15:00:00 | 2018-11-12 12:00:00 | Completed      |
↪  ede837ad   |   7 |  88
36 |       91.98 | 2018-03-28 00:00:00 | 2018-09-29 15:00:00 | 2018-10-05 12:00:00 | Reserved       |
↪  |   5 |  89
37 |       61.46 | 2018-08-22 00:00:00 | 2018-10-23 15:00:00 | 2018-10-28 12:00:00 | Reserved       |
↪  |  58 |  30
38 |       187.2 | 2018-04-30 00:00:00 | 2018-10-14 15:00:00 | 2018-10-22 12:00:00 | Completed      |
↪  9582b7ce   |  82 |  33
39 |      371.27 | 2018-02-02 00:00:00 | 2018-09-21 15:00:00 | 2018-09-27 12:00:00 | Reserved       |
↪  |  71 |  62
40 |      345.02 | 2018-07-05 00:00:00 | 2018-10-31 15:00:00 | 2018-11-08 12:00:00 | Pending        |
↪  |  57 |  65
41 |         391 | 2018-05-14 00:00:00 | 2018-10-06 15:00:00 | 2018-10-15 12:00:00 | Cancelled      |
↪  |  20 |  36
42 |       53.98 | 2018-05-04 00:00:00 | 2018-10-03 15:00:00 | 2018-10-09 12:00:00 | Reserved       |
↪  |  88 |  14
43 |     1225.18 | 2018-08-26 00:00:00 | 2018-11-02 15:00:00 | 2018-11-10 12:00:00 | Void           |
↪  |   6 |  78
44 |      166.97 | 2018-09-17 00:00:00 | 2018-09-23 15:00:00 | 2018-09-28 12:00:00 | Void           |
↪  |  17 |  22
45 |       52.49 | 2018-06-12 00:00:00 | 2018-09-18 15:00:00 | 2018-09-22 12:00:00 | Cancelled      |
↪  |  85 |  18
46 |       59.13 | 2018-06-13 00:00:00 | 2018-09-14 15:00:00 | 2018-09-18 12:00:00 | Pending        |
↪  |  94 |  52
47 |       71.49 | 2018-01-21 00:00:00 | 2018-10-15 15:00:00 | 2018-10-23 12:00:00 | Completed      |
↪  7fc5bf94   |   3 |  16
48 |      272.55 | 2018-06-04 00:00:00 | 2018-11-04 15:00:00 | 2018-11-05 12:00:00 | Void           |
↪  |  55 |  53
49 |       86.09 | 2018-06-06 00:00:00 | 2018-09-27 15:00:00 | 2018-10-05 12:00:00 | Pending        |
↪  |  51 |  34
50 |      202.96 | 2018-06-29 00:00:00 | 2018-11-04 15:00:00 | 2018-11-05 12:00:00 | Reserved       |
↪  |  72 |  56
51 |      189.79 | 2018-05-01 00:00:00 | 2018-09-27 15:00:00 | 2018-10-04 12:00:00 | Cancelled      |
↪  |  19 |  28
52 |        71.9 | 2018-05-07 00:00:00 | 2018-10-28 15:00:00 | 2018-11-04 12:00:00 | Completed      |
↪  dabfa618   |  34 |   5
53 |        1111 | 2018-09-13 00:00:00 | 2018-10-18 15:00:00 | 2018-10-22 12:00:00 | Completed      |
↪  57532c4c   |  67 |  73
54 |      108.65 | 2018-07-13 00:00:00 | 2018-11-04 15:00:00 | 2018-11-11 12:00:00 | Pending        |
↪  |  77 |  59
55 |       373.3 | 2018-05-06 00:00:00 | 2018-10-17 15:00:00 | 2018-10-24 12:00:00 | Void           |
↪  |  45 |  83
56 |       67.56 | 2018-07-10 00:00:00 | 2018-09-28 15:00:00 | 2018-10-04 12:00:00 | Void           |
↪  |  15 |  78
57 |         323 | 2018-05-03 00:00:00 | 2018-10-22 15:00:00 | 2018-10-31 12:00:00 | Completed      |
↪  c5a46f10   |   6 |  67
58 |       50.13 | 2018-09-28 00:00:00 | 2018-11-03 15:00:00 | 2018-11-04 12:00:00 | Cancelled      |
↪  |  97 |  48
59 |      355.92 | 2018-04-06 00:00:00 | 2018-10-23 15:00:00 | 2018-10-25 12:00:00 | Void           |
↪  |  64 |  87
60 |      254.94 | 2018-07-09 00:00:00 | 2018-09-20 15:00:00 | 2018-09-21 12:00:00 | Pending        |
↪  |   3 |  44
61 |      251.06 | 2018-01-28 00:00:00 | 2018-10-22 15:00:00 | 2018-10-23 12:00:00 | Pending        |
↪  |  59 |  43
62 |      230.48 | 2018-07-23 00:00:00 | 2018-09-13 15:00:00 | 2018-09-18 12:00:00 | Pending        |
↪  |  66 |  45
63 |       81.25 | 2018-09-08 00:00:00 | 2018-09-27 15:00:00 | 2018-10-01 12:00:00 | Void           |
↪  |  10 |  42
64 |      143.57 | 2018-09-09 00:00:00 | 2018-10-23 15:00:00 | 2018-10-27 12:00:00 | Cancelled      |
↪  |  80 |  17
65 |      381.09 | 2018-05-06 00:00:00 | 2018-10-06 15:00:00 | 2018-10-09 12:00:00 | Void           |
↪  |  15 |  30
66 |       54.67 | 2018-06-20 00:00:00 | 2018-10-23 15:00:00 | 2018-10-28 12:00:00 | Void           |
↪  |  73 |  74
```

```
 67 |      372.49 | 2018-04-12 00:00:00 | 2018-11-07 15:00:00 | 2018-11-11 12:00:00 | Pending        |
↪  |  51 |  10
 68 |      225.41 | 2018-09-19 00:00:00 | 2018-10-06 15:00:00 | 2018-10-15 12:00:00 | Cancelled      |
↪  |  26 |  97
 69 |      133.99 | 2018-04-25 00:00:00 | 2018-09-13 15:00:00 | 2018-09-14 12:00:00 | Pending        |
↪  |  63 |  91
 70 |      132.37 | 2018-06-17 00:00:00 | 2018-10-21 15:00:00 | 2018-10-22 12:00:00 | Reserved       |
↪  |  87 |  48
 71 |       47.17 | 2018-05-10 00:00:00 | 2018-11-03 15:00:00 | 2018-11-10 12:00:00 | Completed      |
↪  898d792b   |  10 |  50
 72 |       60.89 | 2018-07-15 00:00:00 | 2018-11-02 15:00:00 | 2018-11-04 12:00:00 | Pending        |
↪  |  51 |  96
 73 |       213.8 | 2018-03-21 00:00:00 | 2018-10-07 15:00:00 | 2018-10-15 12:00:00 | Cancelled      |
↪  |  30 |  59
 74 |       79.77 | 2018-05-26 00:00:00 | 2018-10-27 15:00:00 | 2018-10-31 12:00:00 | Void           |
↪  |  12 |  15
 75 |      353.68 | 2018-08-22 00:00:00 | 2018-10-22 15:00:00 | 2018-10-30 12:00:00 | Completed      |
↪  e3d61ddf   |  83 |  52
 76 |      312.29 | 2018-08-15 00:00:00 | 2018-09-17 15:00:00 | 2018-09-21 12:00:00 | Cancelled      |
↪  |  44 |  65
 77 |      259.98 | 2018-08-03 00:00:00 | 2018-09-26 15:00:00 | 2018-09-28 12:00:00 | Void           |
↪  |  33 |   3
 78 |      353.04 | 2018-02-07 00:00:00 | 2018-09-25 15:00:00 | 2018-09-26 12:00:00 | Completed      |
↪  48aaa41f   |  95 | 100
 79 |       84.61 | 2018-05-17 00:00:00 | 2018-11-06 15:00:00 | 2018-11-08 12:00:00 | Void           |
↪  |  69 |  88
 80 |       117.4 | 2018-09-28 00:00:00 | 2018-10-31 15:00:00 | 2018-11-02 12:00:00 | Pending        |
↪  |  69 |  18
 81 |      335.67 | 2018-06-02 00:00:00 | 2018-10-28 15:00:00 | 2018-10-29 12:00:00 | Cancelled      |
↪  |  48 |  91
 82 |       173.6 | 2018-04-01 00:00:00 | 2018-09-17 15:00:00 | 2018-09-26 12:00:00 | Cancelled      |
↪  |  21 |  17
 83 |     1189.65 | 2018-01-16 00:00:00 | 2018-10-25 15:00:00 | 2018-10-31 12:00:00 | Cancelled      |
↪  |  58 |  51
 84 |        99.4 | 2018-09-02 00:00:00 | 2018-09-13 15:00:00 | 2018-09-17 12:00:00 | Cancelled      |
↪  |  49 |   9
 85 |      296.64 | 2018-01-30 00:00:00 | 2018-09-30 15:00:00 | 2018-10-04 12:00:00 | Completed      |
↪  6ecd554b   |  95 |  12
 86 |      372.72 | 2018-03-31 00:00:00 | 2018-10-27 15:00:00 | 2018-10-29 12:00:00 | Reserved       |
↪  |  28 |  19
 87 |      163.33 | 2018-09-01 00:00:00 | 2018-11-03 15:00:00 | 2018-11-08 12:00:00 | Reserved       |
↪  |  34 |  56
 88 |      376.19 | 2018-02-21 00:00:00 | 2018-11-01 15:00:00 | 2018-11-08 12:00:00 | Reserved       |
↪  |   3 |  50
 89 |      141.96 | 2018-04-09 00:00:00 | 2018-09-27 15:00:00 | 2018-10-03 12:00:00 | Void           |
↪  |  29 |  98
 90 |       32.19 | 2018-01-15 00:00:00 | 2018-10-08 15:00:00 | 2018-10-11 12:00:00 | Void           |
↪  |  37 |  31
 91 |       58.44 | 2018-08-13 00:00:00 | 2018-10-14 15:00:00 | 2018-10-21 12:00:00 | Pending        |
↪  |   9 |  12
 92 |      379.53 | 2018-03-16 00:00:00 | 2018-10-03 15:00:00 | 2018-10-10 12:00:00 | Completed      |
↪  ecfc52ec   |   1 |  75
 93 |       37.08 | 2018-01-09 00:00:00 | 2018-09-28 15:00:00 | 2018-10-07 12:00:00 | Pending        |
↪  |  84 |   6
 94 |      213.02 | 2018-06-05 00:00:00 | 2018-10-21 15:00:00 | 2018-10-29 12:00:00 | Reserved       |
↪  |  19 |  38
 95 |       87.88 | 2018-03-23 00:00:00 | 2018-10-20 15:00:00 | 2018-10-22 12:00:00 | Pending        |
↪  |  76 |  25
 96 |       225.9 | 2018-02-01 00:00:00 | 2018-10-26 15:00:00 | 2018-10-30 12:00:00 | Completed      |
↪  fbc19155   |  93 |  89
 97 |      343.79 | 2018-08-14 00:00:00 | 2018-11-06 15:00:00 | 2018-11-10 12:00:00 | Void           |
↪  |  96 |  92
 98 |       115.8 | 2018-08-12 00:00:00 | 2018-09-20 15:00:00 | 2018-09-28 12:00:00 | Pending        |
↪  |  51 |  92
 99 |      170.66 | 2018-09-05 00:00:00 | 2018-09-24 15:00:00 | 2018-10-02 12:00:00 | Completed      |
↪  3051728c   |  13 |  81
100 |      328.82 | 2018-09-16 00:00:00 | 2018-11-04 15:00:00 | 2018-11-05 12:00:00 | Reserved       |
↪  |  22 |  80
```

```
(100 rows)
```

Relation Property_rating - DESCRIBE

```
palms=> \d property_rating
                                        Table "public.property_rating"
   Column    |            Type             | Collation | Nullable |                    Default
-------------+-----------------------------+-----------+----------+-----------------------------------------------
 prid        | integer                     |           | not null |
 ↪  nextval('property_rating_prid_seq'::regclass)
 rating      | integer                     |           |          |
 review      | character varying(1024)     |           |          |
 rating_date | timestamp without time zone |           | not null |
 pid         | integer                     |           |          |
 bid         | integer                     |           |          |
 gid         | integer                     |           |          |
Indexes:
    "property_rating_prid_pk" PRIMARY KEY, btree (prid)
Check constraints:
    "property_rating_rating_check" CHECK (rating >= 1 AND rating <= 5)
Foreign-key constraints:
    "property_rating_bid_fk" FOREIGN KEY (bid) REFERENCES booking(bid)
    "property_rating_gid_fk" FOREIGN KEY (gid) REFERENCES guest(gid)
    "property_rating_pid_fk" FOREIGN KEY (pid) REFERENCES property(pid)
```

SELECT * FROM Propertyrating:

prid	rating	review	rating_date	pid	bid	gid
1	1	pulvinar sed	2018-09-07 00:00:00	60	8	95
2	5		2018-01-26 00:00:00	49	91	91
3	5		2018-02-11 00:00:00	38	17	23
4	4		2017-11-11 00:00:00	21	34	76
5	4	urna	2018-02-02 00:00:00	44	4	70
6	5	cursus	2018-04-05 00:00:00	96	98	97
7	4	molestie lorem	2018-08-24 00:00:00	15	90	25
8	3		2018-10-04 00:00:00	72	96	47
9	4	et ultrices	2018-06-20 00:00:00	42	74	43
10	4		2017-11-17 00:00:00	21	68	8
11	5		2018-03-25 00:00:00	23	76	44
12	2	eget eleifend	2018-06-05 00:00:00	42	95	66
13	3	in	2018-05-16 00:00:00	16	36	4
14	4		2018-05-17 00:00:00	56	19	30
15	5		2018-01-21 00:00:00	55	2	63
16	4		2017-11-11 00:00:00	83	51	82
17	3		2018-01-27 00:00:00	91	89	83
18	3	non	2017-12-29 00:00:00	56	53	45
19	2		2018-02-24 00:00:00	33	8	75
20	4		2018-05-26 00:00:00	65	73	21
21	2	aliquam lacus	2018-03-11 00:00:00	87	13	59
22	4		2018-07-25 00:00:00	7	96	88
23	2	rutrum rutrum	2018-10-02 00:00:00	29	33	27
24	2	vel	2017-11-15 00:00:00	84	78	54
25	4		2018-02-06 00:00:00	36	27	22
26	5		2018-03-24 00:00:00	77	92	56
27	4	nisi eu	2018-04-24 00:00:00	76	100	6
28	4	in	2018-09-22 00:00:00	88	83	36
29	4	dui	2018-04-03 00:00:00	85	25	13
30	4		2018-05-09 00:00:00	87	31	84
31	4		2018-04-25 00:00:00	87	61	70
32	5		2018-01-21 00:00:00	33	65	87
33	2	vestibulum	2018-04-26 00:00:00	42	56	3
34	1		2018-02-08 00:00:00	52	73	32
35	3		2018-04-05 00:00:00	87	66	73
36	3	integer ac	2017-11-03 00:00:00	75	70	89

```
  37 |      3 |                | 2017-11-24 00:00:00 |  70 |  81 |  84
  38 |      4 |                | 2018-08-15 00:00:00 |  18 |  68 |  59
  39 |      4 | nulla nisl     | 2018-04-26 00:00:00 |  49 |  82 |  36
  40 |      2 |                | 2018-07-26 00:00:00 |  15 |  23 |  68
  41 |      4 |                | 2017-11-19 00:00:00 |   9 |  36 |  21
  42 |      4 |                | 2017-12-29 00:00:00 |   3 |  33 |  50
  43 |      2 | a nibh         | 2018-07-22 00:00:00 |  87 |  96 |  36
  44 |      2 | vitae nisl     | 2017-10-31 00:00:00 |  81 |  65 |  71
  45 |      4 |                | 2018-08-14 00:00:00 |  46 |  90 |   9
  46 |      2 |                | 2018-07-11 00:00:00 |  39 |  72 |  35
  47 |      4 | sit            | 2018-06-09 00:00:00 |  21 |  77 |  62
  48 |      4 |                | 2018-01-12 00:00:00 |   7 |  63 |  60
  49 |      4 | vivamus        | 2017-10-22 00:00:00 |  90 |  65 |  76
  50 |      1 |                | 2018-03-06 00:00:00 |  33 |  78 |  94
  51 |      2 | in est         | 2018-02-27 00:00:00 |  37 |  54 |  75
  52 |      5 | rhoncus sed    | 2018-09-11 00:00:00 |  78 |   7 |  73
  53 |      1 |                | 2017-11-12 00:00:00 |  19 |  34 |  88
  54 |      4 | donec odio     | 2018-07-07 00:00:00 |   8 |  19 |  69
  55 |      3 |                | 2017-12-13 00:00:00 |   6 |   2 |  63
  56 |      3 | ac leo         | 2018-08-04 00:00:00 |  80 |  93 |   4
  57 |      2 |                | 2018-05-13 00:00:00 |  52 |  42 |   9
  58 |      2 | donec posuere  | 2018-09-04 00:00:00 |  68 |  67 |  75
  59 |      4 |                | 2018-09-05 00:00:00 |  27 |  63 |  64
  60 |      4 | enim           | 2018-05-25 00:00:00 |  80 |  69 |  75
  61 |      4 | feugiat et     | 2018-09-24 00:00:00 |  43 |  71 |  21
  62 |      2 |                | 2018-09-11 00:00:00 |  61 |  73 |  20
  63 |      2 |                | 2018-04-23 00:00:00 |  58 |  80 |   4
  64 |      4 |                | 2017-12-30 00:00:00 |  43 |  29 |  25
  65 |      2 | dui            | 2018-05-18 00:00:00 |   4 |  34 |  84
  66 |      4 | metus          | 2018-08-31 00:00:00 |  66 |   3 |  69
  67 |      4 | est congue     | 2018-05-01 00:00:00 |  98 |  37 |  28
  68 |      1 |                | 2018-08-29 00:00:00 |  15 |  44 |  11
  69 |      4 | est            | 2018-10-18 00:00:00 |  13 |  67 |  81
  70 |      3 | felis donec    | 2018-05-30 00:00:00 |  51 |  97 |  28
  71 |      4 | auctor gravida | 2017-12-04 00:00:00 |  61 |  50 |  86
  72 |      3 | nulla ut       | 2018-07-29 00:00:00 |  57 |  79 |  41
  73 |      1 |                | 2018-04-09 00:00:00 |  70 |  20 |   5
  74 |      2 |                | 2018-03-29 00:00:00 |  19 |  83 |  12
  75 |      1 | semper         | 2018-07-15 00:00:00 |  42 |  62 |  55
  76 |      5 | orci vehicula  | 2017-12-13 00:00:00 |  34 | 100 |  21
  77 |      3 | in lectus      | 2018-08-29 00:00:00 |  28 |  82 |  79
  78 |      1 | at nibh        | 2018-06-23 00:00:00 |  34 |  11 |   6
  79 |      4 | lobortis sapien | 2018-03-02 00:00:00 |   1 |  75 |  18
  80 |      3 | lacus          | 2017-10-30 00:00:00 |  17 |  21 |  40
  81 |      5 |                | 2017-10-08 00:00:00 |  46 |  30 |  50
  82 |      3 |                | 2018-10-23 00:00:00 |  12 |   1 |  15
  83 |      4 |                | 2018-05-18 00:00:00 |  80 |  91 |  54
  84 |      3 | in sapien      | 2018-08-07 00:00:00 |  90 |  62 |  45
  85 |      1 | et ultrices    | 2018-04-25 00:00:00 |  68 |  84 |  36
  86 |      4 |                | 2017-10-24 00:00:00 |  80 |   9 |  11
  87 |      5 | lacinia        | 2018-08-27 00:00:00 |  94 |  92 |  64
  88 |      3 |                | 2017-10-06 00:00:00 |  40 |  21 |  68
  89 |      4 |                | 2018-04-19 00:00:00 |   5 |  88 |  99
  90 |      1 | convallis nunc | 2018-04-17 00:00:00 |  58 |  63 |   4
  91 |      2 | urna ut        | 2018-04-22 00:00:00 |  67 |  78 |  30
  92 |      4 | maecenas ut    | 2018-01-05 00:00:00 |  52 |  82 |  26
  93 |      4 | mattis         | 2018-02-08 00:00:00 |  72 |  30 |   2
  94 |      1 | lobortis vel   | 2018-01-16 00:00:00 |  26 |  56 |  37
  95 |      4 | nec            | 2018-02-19 00:00:00 |  10 |  27 |  20
  96 |      1 | in felis       | 2017-10-25 00:00:00 |  96 |   7 |  92
  97 |      5 |                | 2017-12-03 00:00:00 |   4 |  25 |  27
  98 |      4 |                | 2018-07-13 00:00:00 |  56 |  93 |  46
  99 |      4 | nunc rhoncus   | 2017-12-14 00:00:00 |  54 |  55 |  46
 100 |      4 | interdum       | 2018-08-28 00:00:00 |  18 |  60 |  69
(100 rows)
```

Relation Address - DESCRIBE

```
palms=> \d address
                                    Table "public.address"
   Column    |         Type          | Collation | Nullable |               Default
-------------+-----------------------+-----------+----------+-------------------------------------
 aid         | integer               |           | not null | nextval('address_aid_seq'::regclass)
 street      | character varying(64) |           |          |
 city        | character varying(64) |           |          |
 postal_code | character varying(16) |           |          |
 country     | character varying(64) |           |          |
Indexes:
    "address_pk" PRIMARY KEY, btree (aid)
Check constraints:
    "address_city_check" CHECK (length(city::text) >= 1)
    "address_country_check" CHECK (length(country::text) >= 1)
    "address_postal_code_check" CHECK (length(postal_code::text) >= 1)
    "address_street_check" CHECK (length(street::text) >= 1)
Referenced by:
    TABLE "includes" CONSTRAINT "includes_aid_fk" FOREIGN KEY (aid) REFERENCES address(aid)
    TABLE "livesat" CONSTRAINT "livesat_aid_fk" FOREIGN KEY (aid) REFERENCES address(aid)
    TABLE "operatesat" CONSTRAINT "operatesat_aid_fk" FOREIGN KEY (aid) REFERENCES address(aid)
```

SELECT * FROM Address:

aid	street	city	postal_code	country
1	9 Gerald Drive	San Juan	56030	Mexico
2	5931 Grim Lane	Santa Maria	51009	Mexico
3	21 Calypso Parkway	Virginia Beach	23459	United States
4	00485 Shopko Road	Shreveport	71151	United States
5	99273 Lunder Center	Wilkes Barre	18763	United States
6	6 Huxley Alley	Reston	20195	United States
7	360 Claremont Drive	Sainte-Thérèse	J7G	Canada
8	90590 Blue Bill Park Way	Altona	K1W	Canada
9	6 Northport Lane	Labrador City	A2V	Canada
10	945 Hovde Circle	Baie-Saint-Paul	G3Z	Canada
11	00668 Mockingbird Street	Tyler	75799	United States
12	094 Butterfield Alley	Emiliano Zapata	32883	Mexico
13	Elm Street	Las Vegas	89135	United States
14	3140 Spaight Circle	La Huerta	53338	Mexico
15	3656 Mockingbird Place	Bloomington	47405	United States
16	59816 Colorado Terrace	Kansas City	66112	United States
17	553 Di Loreto Avenue	Middleton	LE16	United Kingdom
18	64955 Schiller Center	Lubbock	79491	United States
19	5 Homewood Avenue	Dallas	75205	United States
20	6 Hayes Plaza	Atlanta	31165	United States
21	134 Lukken Drive	Norfolk	23504	United States
22	538 Marquette Avenue	Sacramento	94250	United States
23	4 Blackbird Drive	San Cristobal	36800	Mexico
24	4 Rutledge Junction	5 de Mayo	87785	Mexico
25	40350 Farwell Alley	Renfrew	K7V	Canada
26	62123 Portage Point	Milwaukee	53234	United States
27	2509 Marcy Circle	Revelstoke	H9K	Canada
28	65 Old Shore Plaza	Guadalupe Victoria	87086	Mexico
29	0716 Elm Street	Paterson	07505	United States
30	86487 Mayfield Park	Magrath	J7G	Canada
31	6 Briar Crest Place	Lubbock	79405	United States
32	1 Oakridge Junction	Newton	IV1	United Kingdom
33	76 Bayside Center	San Francisco	95710	United States
34	095 Crowley Hill	El Mirador	56564	Mexico
35	85885 Eggendart Trail	Killam	A0A	Canada
36	99 Tennyson Lane	Sacramento	95852	United States
37	65566 Eliot Center	Montgomery	36195	United States
38	1331 International Terrace	El Calvario	68213	Mexico
39	1222 Banding Pass	San Antonio	78220	United States
40	35886 Lakeland Place	Pueblo	81010	United States
41	11 Becker Place	Venustiano Carranza	93848	Mexico
42	61859 Northwestern Alley	San Isidro	60284	Mexico
43	7 Montana Parkway	Benito Juarez	91716	Mexico

```
  44 | 1722 Division Court        | Benito Juarez       | 41800       | Mexico
  45 | 03687 Ridge Oak Court      | Saint-Sauveur       | J0R         | Canada
  46 | 913 East Drive             | Lanigan             | J7A         | Canada
  47 | 6177 Schlimgen Point       | Santa Cruz          | 30560       | Mexico
  48 | 32040 Red Cloud Center     | Hollywood           | 33023       | United States
  49 | 70991 Red Cloud Way        | Shediac             | E4P         | Canada
  50 | 4 Anderson Avenue          | Nashville           | 37205       | United States
  51 | 682 Prentice Parkway       | Houston             | 77293       | United States
  52 | 630 Cottonwood Parkway     | San Francisco       | 78490       | United States
  53 | 83973 Gina Circle          | Lawrenceville       | 30245       | United States
  54 | 61698 Fair Oaks Center     | Vista Hermosa       | 39130       | Mexico
  55 | 5 Toban Court              | New York City       | 10060       | United States
  56 | 933 Jenifer Parkway        | La Concepcion       | 52105       | Mexico
  57 | 52142 Aberg Drive          | Providence          | 02912       | United States
  58 | 62325 Butterfield Point    | Columbus            | 43220       | United States
  59 | 6 Elm Street               | Dallas              | 75287       | United States
  60 | 2398 Village Green Drive   | Lindavista          | 70633       | Mexico
  61 | 18 Surrey Court            | Camlachie           | G5Z         | Canada
  62 | 20672 Weeping Birch Place  | Santa Elena         | 30807       | Mexico
  63 | 3 Ruskin Avenue            | Iowa City           | 52245       | United States
  64 | 6824 Esch Alley            | La Victoria         | 95603       | Mexico
  65 | 78918 Farwell Alley        | Stockton            | 95205       | United States
  66 | 50 Elka Point              | Donnacona           | G3M         | Canada
  67 | 858 Novick Point           | Lac La Biche        | E1W         | Canada
  68 | 92637 Main Plaza           | Saint Louis         | 63180       | United States
  69 | 681 Crowley Lane           | Orlando             | 32819       | United States
  70 | 0312 Village Drive         | Des Moines          | 50393       | United States
  71 | 55349 Green Ridge Parkway  | Monroe              | 71213       | United States
  72 | 5006 Lotheville Road       | Twyford             | LE14        | United Kingdom
  73 | Elm Street                 | Saguenay            | G7K         | Canada
  74 | 07 Lighthouse Bay Avenue   | El Salitre          | 58503       | Mexico
  75 | 4605 2nd Point             | Oklahoma City       | 73142       | United States
  76 | 60281 Clarendon Drive      | El Limon            | 92105       | Mexico
  77 | 7 Maple Wood Point         | Vista Hermosa       | 39130       | Mexico
  78 | 381 Holmberg Park          | Sutton              | CT15        | United Kingdom
  79 | 07 Golf Court              | Houston             | 77293       | United States
  80 | 9 Nancy Plaza              | Cuauhtemoc          | 40068       | Mexico
  81 | 23 Gale Drive              | Juan N Alvarez      | 39030       | Mexico
  82 | 094 Jenifer Alley          | Pasadena            | B2J         | Canada
  83 | 8 Daystar Road             | Tacoma              | 98405       | United States
  84 | 50730 Glacier Hill Avenue  | Parrsboro           | L2A         | Canada
  85 | 30 Warrior Road            | 20 de Noviembre     | 93828       | Mexico
  86 | 37 Ruskin Circle           | Mesa                | 85205       | United States
  87 | 0800 Westend Pass          | San Agustin         | 29480       | Mexico
  88 | 17 Mcguire Alley           | Kensington          | S7K         | Canada
  89 | 423 Sycamore Crossing      | Jonquière           | G7Y         | Canada
  90 | 98477 Coleman Place        | Wilkes Barre        | 18706       | United States
  91 | 248 Becker Alley           | Detroit             | 48232       | United States
  92 | 142 Mandrake Center        | Denton              | M34         | United Kingdom
  93 | 71 Cottonwood Terrace      | Detroit             | 48217       | United States
  94 | 45281 Sage Junction        | Boise               | 83722       | United States
  95 | 04794 Reindahl Parkway     | Burgeo              | N9A         | Canada
  96 | 61404 Ohio Lane            | Denton              | 76210       | United States
  97 | 92 Sachtjen Point          | Evansville          | 47737       | United States
  98 | 73478 Hoard Point          | Washington          | 20409       | United States
  99 | 031 Kenwood Center         | La Laguna           | 51247       | Mexico
 100 | 674 Trailsway Court        | Port Colborne       | L3K         | Canada
(100 rows)
```

Relation Phone - DESCRIBE

```
palms=> \d phone
                                  Table "public.phone"
    Column    |         Type          | Collation | Nullable |              Default
--------------+-----------------------+-----------+----------+-----------------------------------
 phid         | integer               |           | not null | nextval('phone_phid_seq'::regclass)
```

```
 phone_number | character varying(64) |           |          |
 phone_type   | character varying(64) |           | not null |
 gid          | integer               |           |          |
 oid          | integer               |           |          |
Indexes:
    "phone_id_pk" PRIMARY KEY, btree (phid)
Check constraints:
    "phone_phone_number_check" CHECK (length(phone_number::text) >= 7)
Foreign-key constraints:
    "phone_gid_fk" FOREIGN KEY (gid) REFERENCES guest(gid)
    "phone_oid_fk" FOREIGN KEY (oid) REFERENCES owner(oid)
```

SELECT * FROM Phone:

phid	phone_number	phone_type	gid	oid
1	6707702050	Home	55	
2	2686847640	Mobile	85	
3	8974034405	Other	84	
4	2146612197	Work		6
5	9971786818	Other	13	
6	2481153325	Mobile		85
7	2073487356	Mobile	16	
8	9289593733	Home	23	
9	4603114742	Other	92	
10	5332914896	Work	11	
11	9573829756	Work		56
12	6983811143	Other		40
13	5587305080	Mobile		42
14	9663460601	Other	81	
15	1469420580	Other		74
16	8463110224	Mobile		4
17	8346294971	Mobile	38	
18	2124979385	Home	71	
19	4724295763	Mobile	36	
20	6138366275	Mobile	59	
21	1049587236	Work	33	
22	4216191521	Home		35
23	4672124625	Home	15	
24	2673538146	Mobile	50	
25	6948470147	Home	75	
26	2422004719	Other	88	
27	1881382322	Mobile	70	
28	3394661197	Work	14	
29	9715823661	Other	32	
30	6882792580	Home		65
31	7306044682	Other	71	
32	7732856172	Home	70	
33	4299003047	Mobile		37
34	4249994840	Mobile	41	45
35	2407955416	Work	85	
36	8773051509	Mobile		66
37	7456245618	Mobile		48
38	9971786818	Other	13	
39	2328903472	Other	95	
40	1981422863	Other	1	
41	3739452930	Work	1	
42	9106768111	Other	8	
43	4239539513	Work		1
44	7856568262	Work	53	
45	9193491665	Work	18	
46	9554274795	Other		85
47	9167299711	Work		51
48	1771462251	Other	99	
49	7179464496	Other		4
50	9422281745	Home	43	
51	6934051654	Other	23	
52	5866960308	Other		36

```
  53 | 3037695904   | Mobile     |     |  14
  54 | 6162317636   | Work       |  72 |
  55 | 9422281745   | Home       |  43 |
  56 | 8983553590   | Home       |  70 |
  57 | 6804858994   | Home       |  13 |
  58 | 1724067554   | Home       |  50 |
  59 | 9185386923   | Home       |     |  18
  60 | 6948470147   | Home       |  75 |
  61 | 1153680394   | Other      |  97 |
  62 | 9289593733   | Home       |  23 |
  63 | 1188240110   | Work       |     |  64
  64 | 8658936979   | Mobile     |  65 |
  65 | 2615749244   | Home       |     |  41
  66 | 1805169564   | Other      |   4 |
  67 | 7688747867   | Mobile     |     |  29
  68 | 4231881146   | Other      |  46 |
  69 | 8149205447   | Work       |     |  54
  70 | 8093015440   | Home       |  27 |
  71 | 2422004719   | Other      |  88 |
  72 | 5466273238   | Home       |     |  56
  73 | 2806696188   | Work       |  62 |
  74 | 6948635253   | Other      |  27 |
  75 | 2938846463   | Work       |     |  82
  76 | 7597066270   | Other      |     |  37
  77 | 9603693677   | Home       |     |  45
  78 | 9653772608   | Other      |  80 |
  79 | 4684570922   | Home       |  51 |
  80 | 2957802922   | Mobile     |     |  31
  81 | 6804858994   | Home       |  13 |
  82 | 3929915928   | Work       |  80 |
  83 | 4341369139   | Other      |   1 |
  84 | 4263137593   | Other      |   5 |
  85 | 2331938221   | Home       |   8 |
  86 | 4616097395   | Work       |     |   7
  87 | 1533058114   | Mobile     |  10 |
  88 | 7298288809   | Work       |     |   4
  89 | 7725896083   | Other      |  10 |
  90 | 5555567698   | Mobile     |     |  12
  91 | 1049587236   | Work       |  33 |
  92 | 9351445175   | Work       |  26 |
  93 | 3739452930   | Work       |   1 |
  94 | 6707702050   | Home       |  55 |
  95 | 4206251620   | Other      |     |  18
  96 | 8586450411   | Home       |  55 |
  97 | 9106768111   | Other      |   8 |
  98 | 8923838292   | Other      |  27 |
  99 | 4131613725   | Work       |  87 |
 100 | 3528204391   | Work       |     |  91
(100 rows)
```

Relation Offers - DESCRIBE

```
palms=> \d offers
                        Table "public.offers"
   Column   |            Type             | Collation | Nullable | Default
------------+-----------------------------+-----------+----------+---------
 oid        | integer                     |           |          |
 pid        | integer                     |           |          |
 start_date | timestamp without time zone |           | not null | now()
 end_date   | timestamp without time zone |           |          |
Foreign-key constraints:
    "offers_oid_fk" FOREIGN KEY (oid) REFERENCES owner(oid)
    "offers_pid_fk" FOREIGN KEY (pid) REFERENCES property(pid)
```

SELECT * FROM Offers:

```
 oid | pid |     start_date      |      end_date
-----+-----+---------------------+---------------------
  55 |  18 | 2018-08-30 00:00:00 |
   5 |  84 | 2018-05-29 00:00:00 |
  84 |  72 | 2018-01-23 00:00:00 |
  20 |  97 | 2018-08-10 00:00:00 |
  82 |  35 | 2018-04-13 00:00:00 |
   9 |  97 | 2018-09-08 00:00:00 |
  95 |  86 | 2017-12-03 00:00:00 |
  89 |  80 | 2017-11-29 00:00:00 |
  40 |  70 | 2018-03-23 00:00:00 |
  31 |  34 | 2018-07-29 00:00:00 |
  59 |   8 | 2018-06-29 00:00:00 |
  37 |  73 | 2018-07-09 00:00:00 |
  11 |  23 | 2018-08-25 00:00:00 |
  91 |  13 | 2018-04-10 00:00:00 |
  63 |  20 | 2018-02-05 00:00:00 |
  81 |  60 | 2018-05-22 00:00:00 |
  38 |  69 | 2018-10-08 00:00:00 | 2019-04-03 00:00:00
  42 |   5 | 2018-05-19 00:00:00 |
  77 |   5 | 2018-06-21 00:00:00 |
  86 |   8 | 2018-03-09 00:00:00 |
  67 |  36 | 2017-11-05 00:00:00 |
  87 |  96 | 2018-02-23 00:00:00 |
  46 |  46 | 2018-07-15 00:00:00 |
  18 |  69 | 2018-08-04 00:00:00 |
  81 |  12 | 2018-09-08 00:00:00 |
  31 |  95 | 2018-03-25 00:00:00 |
  54 |  38 | 2018-06-01 00:00:00 |
   8 |  35 | 2018-08-05 00:00:00 |
  84 |  92 | 2018-03-08 00:00:00 | 2018-12-18 00:00:00
  25 |  96 | 2018-02-23 00:00:00 |
  21 |  89 | 2018-04-07 00:00:00 |
  22 |  28 | 2018-05-12 00:00:00 | 2018-10-01 00:00:00
  20 |  63 | 2018-03-13 00:00:00 |
   9 |  43 | 2018-03-17 00:00:00 |
  87 |  37 | 2018-02-12 00:00:00 |
  69 |  65 | 2018-10-10 00:00:00 |
  88 |   6 | 2017-11-13 00:00:00 |
  96 |  13 | 2018-02-04 00:00:00 | 2017-10-21 00:00:00
  59 |   9 | 2018-02-27 00:00:00 |
  58 |  38 | 2018-07-29 00:00:00 |
  55 |  84 | 2018-04-15 00:00:00 |
  91 |  76 | 2018-01-14 00:00:00 |
  97 |  62 | 2018-02-27 00:00:00 | 2019-05-05 00:00:00
  20 |  29 | 2018-05-09 00:00:00 |
  70 |  88 | 2018-07-19 00:00:00 |
  49 |  14 | 2018-01-22 00:00:00 |
  16 |   9 | 2018-03-20 00:00:00 |
  47 |  30 | 2018-06-04 00:00:00 |
  96 |  90 | 2018-02-03 00:00:00 |
  19 |  62 | 2018-08-22 00:00:00 |
  28 |  15 | 2017-11-01 00:00:00 |
  89 |  82 | 2018-09-04 00:00:00 |
  51 |  14 | 2018-06-29 00:00:00 |
  97 |  36 | 2018-01-30 00:00:00 |
  17 |  23 | 2018-02-08 00:00:00 |
  83 |  79 | 2018-06-15 00:00:00 |
  32 |  69 | 2018-05-18 00:00:00 |
  59 |  99 | 2018-03-02 00:00:00 |
   1 |  86 | 2018-10-11 00:00:00 |
  66 |  29 | 2018-05-28 00:00:00 |
  71 |  82 | 2018-03-25 00:00:00 |
  53 |  56 | 2018-02-27 00:00:00 |
   6 |  19 | 2018-09-20 00:00:00 |
  42 |  97 | 2018-09-20 00:00:00 |
  34 |  73 | 2018-01-24 00:00:00 |
  16 |  93 | 2018-05-19 00:00:00 |
```

```
  73 |  86 | 2018-07-07 00:00:00 |
  33 |  12 | 2018-01-16 00:00:00 |
   3 |  67 | 2018-01-01 00:00:00 |
  78 |  59 | 2018-04-11 00:00:00 | 2019-10-14 00:00:00
  73 |  50 | 2018-04-09 00:00:00 |
   2 |  74 | 2017-12-07 00:00:00 |
  88 |  50 | 2017-10-31 00:00:00 |
  82 |  24 | 2018-01-24 00:00:00 | 2019-08-20 00:00:00
  49 |  58 | 2017-10-30 00:00:00 |
  47 |  75 | 2017-11-09 00:00:00 | 2018-07-17 00:00:00
   2 |  63 | 2018-05-14 00:00:00 |
  18 |  96 | 2018-08-17 00:00:00 |
  55 |  75 | 2018-06-10 00:00:00 |
  99 |  47 | 2018-08-04 00:00:00 |
  75 |   2 | 2018-01-13 00:00:00 | 2018-10-08 00:00:00
  44 |  49 | 2018-07-03 00:00:00 |
  84 |   9 | 2018-05-17 00:00:00 |
  63 |   1 | 2018-03-09 00:00:00 |
  69 |  14 | 2018-06-01 00:00:00 |
  49 |  15 | 2018-01-01 00:00:00 |
  82 |  24 | 2018-05-14 00:00:00 |
  55 |  71 | 2018-04-12 00:00:00 |
  49 |  35 | 2017-12-24 00:00:00 | 2018-10-26 00:00:00
  49 |  77 | 2018-01-09 00:00:00 |
  99 |   7 | 2018-06-23 00:00:00 |
  90 |  40 | 2018-01-29 00:00:00 | 2018-07-16 00:00:00
   3 |  50 | 2018-09-21 00:00:00 |
  73 |  97 | 2018-04-22 00:00:00 |
  64 |  76 | 2018-01-19 00:00:00 |
  14 |  38 | 2018-03-06 00:00:00 |
  48 |  51 | 2018-04-11 00:00:00 |
  91 |   4 | 2018-08-19 00:00:00 |
  23 |  70 | 2018-06-26 00:00:00 |
  96 |  80 | 2018-09-14 00:00:00 |
(100 rows)
```

Relation Includes - DESCRIBE

```
palms=> \d includes
              Table "public.includes"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 pid    | integer |           | not null |
 aid    | integer |           | not null |
Indexes:
    "includes_pid_aid_fk" PRIMARY KEY, btree (pid, aid)
Foreign-key constraints:
    "includes_aid_fk" FOREIGN KEY (aid) REFERENCES address(aid)
    "includes_pid_fk" FOREIGN KEY (pid) REFERENCES property(pid)
```

SELECT * FROM Includes:

```
 pid | aid
-----+-----
  52 |  77
  36 |  14
  30 |  95
  67 |  67
  18 |  70
 100 |   5
  42 |  78
  42 |  54
  41 |  98
  34 |  52
  98 |  33
```

```
  16 |  52
  14 |  88
  96 |   8
  83 |  22
  98 |  27
   5 |   7
  45 |  74
   4 |  27
  83 |  27
  99 |  77
  82 |  14
  31 |  55
  88 |  96
  22 |  34
  20 |  98
  42 |  60
  88 |  81
  22 |  49
  27 |  47
  89 |  53
   2 |  95
  90 |  91
  38 |  14
  24 |  51
  58 |  30
  28 |  13
  86 |   5
  59 |  39
  68 |  91
  56 |  73
  75 |  16
  36 |  93
  16 |  84
  66 |  32
  42 |  34
  75 |  14
  19 |  22
  90 |  73
  55 |  84
(50 rows)
```

Relation Selects - DESCRIBE

```
palms=> \d selects
              Table "public.selects"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 bid    | integer |           | not null |
 pid    | integer |           | not null |
Indexes:
    "selects_bid_pid_pk" PRIMARY KEY, btree (bid, pid)
Foreign-key constraints:
    "selects_bid_fk" FOREIGN KEY (bid) REFERENCES booking(bid)
    "selects_pid_fk" FOREIGN KEY (pid) REFERENCES property(pid)
```

SELECT * FROM Selects:

```
 bid | pid
-----+-----
  16 |  56
  81 |   8
  34 |  67
  85 |  53
  96 |  26
   5 |  92
```

```
  77 |  22
  58 |  47
  50 |  73
  74 |  32
  80 |  52
  13 |  89
  89 |  90
  83 |  92
  61 |   9
  92 |  89
  16 |  88
  78 |  76
  37 |  51
  77 |  90
  33 |  61
  21 |  16
  66 |  88
   7 |  66
  35 |   5
  38 |  71
  47 |  22
  20 |  69
  89 |  20
  62 |  38
  75 |  30
  98 |  48
   6 |   9
   8 |  67
  71 |  34
  82 |  91
  59 |  86
  70 | 100
  40 |  85
  59 |  56
  78 |  59
  81 |  11
  75 |  53
  65 |  49
  66 |  95
   2 |  56
  46 |  13
  68 |   9
  15 |  33
  66 |  39
(50 rows)
```

Relation Livesat - DESCRIBE

```
palms=> \d livesat
                        Table "public.livesat"
   Column   |            Type             | Collation | Nullable | Default
------------+-----------------------------+-----------+----------+---------
 gid        | integer                     |           | not null |
 aid        | integer                     |           | not null |
 start_date | timestamp without time zone |           | not null | now()
 end_date   | timestamp without time zone |           |          |
Indexes:
    "livesat_gid_aid_pk" PRIMARY KEY, btree (gid, aid)
Foreign-key constraints:
    "livesat_aid_fk" FOREIGN KEY (aid) REFERENCES address(aid)
    "livesat_gid_fk" FOREIGN KEY (gid) REFERENCES guest(gid)
```

SELECT * FROM Livesat:

```
 gid | aid |     start_date      |      end_date
-----+-----+---------------------+---------------------
```

```
60 |  30 | 2018-06-22 00:00:00 | 2018-08-05 00:00:00
85 |  80 | 2018-09-19 00:00:00 |
58 |  53 | 2018-07-10 00:00:00 |
56 |  44 | 2018-04-16 00:00:00 |
76 |  42 | 2018-02-25 00:00:00 |
99 |  59 | 2018-06-12 00:00:00 |
26 |  62 | 2017-11-25 00:00:00 |
30 |  35 | 2018-09-20 00:00:00 |
35 |  17 | 2018-10-08 00:00:00 |
58 |  11 | 2017-11-27 00:00:00 | 2017-12-04 00:00:00
78 |  81 | 2018-01-01 00:00:00 | 2018-06-23 00:00:00
20 |  31 | 2018-09-07 00:00:00 |
62 |  42 | 2018-06-07 00:00:00 |
30 |  15 | 2017-11-12 00:00:00 |
10 |  76 | 2018-08-08 00:00:00 |
93 |  68 | 2018-02-10 00:00:00 | 2018-06-09 00:00:00
10 |  55 | 2018-08-17 00:00:00 |
97 |  63 | 2017-12-07 00:00:00 |
18 |  74 | 2018-09-14 00:00:00 | 2018-10-04 00:00:00
83 |  67 | 2018-08-05 00:00:00 |
31 |  26 | 2018-05-14 00:00:00 |
57 |  20 | 2018-01-08 00:00:00 |
17 |  56 | 2018-07-26 00:00:00 |
73 |  70 | 2017-11-09 00:00:00 | 2018-06-14 00:00:00
 6 |  26 | 2018-01-16 00:00:00 |
33 |  99 | 2018-03-12 00:00:00 |
26 |  58 | 2018-09-27 00:00:00 | 2018-07-05 00:00:00
89 |  79 | 2017-11-18 00:00:00 |
60 |  50 | 2018-05-20 00:00:00 |
78 |  33 | 2018-05-14 00:00:00 |
83 |  70 | 2018-09-17 00:00:00 | 2018-06-28 00:00:00
21 |  57 | 2018-03-19 00:00:00 |
77 |  56 | 2018-04-24 00:00:00 |
48 |  71 | 2018-09-14 00:00:00 |
91 |  22 | 2018-09-23 00:00:00 | 2017-12-23 00:00:00
 5 | 100 | 2018-05-15 00:00:00 |
 8 |  89 | 2018-07-11 00:00:00 |
53 |  77 | 2018-08-04 00:00:00 | 2018-03-16 00:00:00
32 |   4 | 2018-05-21 00:00:00 | 2018-06-29 00:00:00
10 |   5 | 2018-08-17 00:00:00 |
66 |  19 | 2017-11-29 00:00:00 | 2018-08-18 00:00:00
79 |  19 | 2018-08-18 00:00:00 | 2017-11-20 00:00:00
63 |  95 | 2018-10-08 00:00:00 |
50 |  30 | 2018-05-19 00:00:00 |
21 |  34 | 2018-02-19 00:00:00 |
89 |  98 | 2018-02-16 00:00:00 |
57 |  82 | 2018-08-20 00:00:00 |
73 |  47 | 2018-03-04 00:00:00 | 2018-02-09 00:00:00
 6 |  88 | 2018-05-06 00:00:00 |
 5 |  91 | 2018-09-09 00:00:00 |
74 |  15 | 2017-12-05 00:00:00 |
80 |  69 | 2018-09-27 00:00:00 |
47 |  40 | 2018-05-30 00:00:00 |
57 |   5 | 2018-06-11 00:00:00 | 2018-04-26 00:00:00
18 |  60 | 2018-07-22 00:00:00 |
40 |  64 | 2018-02-16 00:00:00 |
65 |  97 | 2018-02-04 00:00:00 |
55 |  20 | 2018-06-17 00:00:00 |
14 |  39 | 2018-05-11 00:00:00 |
51 |  41 | 2018-06-14 00:00:00 |
18 |  15 | 2018-08-12 00:00:00 |
46 |  43 | 2018-05-28 00:00:00 |
77 |  47 | 2018-10-13 00:00:00 |
74 |  21 | 2017-12-12 00:00:00 |
60 |  21 | 2018-06-28 00:00:00 |
21 |  79 | 2018-03-31 00:00:00 | 2017-11-15 00:00:00
50 |  28 | 2018-05-14 00:00:00 | 2017-11-04 00:00:00
77 |  90 | 2018-03-12 00:00:00 |
```

```
  7 |  78 | 2018-02-14 00:00:00 |
  3 |   5 | 2018-03-21 00:00:00 |
 59 |  31 | 2018-07-09 00:00:00 |
 79 |  12 | 2018-01-23 00:00:00 |
 48 |  69 | 2018-09-25 00:00:00 | 2017-12-07 00:00:00
 78 |  77 | 2018-06-22 00:00:00 |
 48 |  30 | 2018-01-10 00:00:00 |
 29 |  78 | 2018-07-11 00:00:00 | 2018-10-18 00:00:00
 88 |  48 | 2018-03-04 00:00:00 | 2017-12-12 00:00:00
 57 |  55 | 2017-11-10 00:00:00 |
  3 |  91 | 2018-04-08 00:00:00 | 2017-11-23 00:00:00
 13 |  25 | 2018-05-28 00:00:00 |
 17 |  65 | 2018-03-21 00:00:00 |
 58 |  13 | 2018-03-23 00:00:00 |
100 |  67 | 2018-09-12 00:00:00 |
 73 |  13 | 2017-11-25 00:00:00 |
 65 |  54 | 2018-02-06 00:00:00 |
 37 |  53 | 2018-02-01 00:00:00 |
 92 |  22 | 2018-06-07 00:00:00 |
 59 |   5 | 2018-06-01 00:00:00 |
 86 |  57 | 2017-12-21 00:00:00 |
 33 |   4 | 2018-06-18 00:00:00 | 2018-07-17 00:00:00
 81 |  88 | 2018-02-22 00:00:00 |
 42 |  77 | 2018-01-31 00:00:00 |
 72 |  20 | 2018-02-18 00:00:00 |
  1 |  57 | 2017-10-29 00:00:00 |
 91 |  29 | 2018-09-22 00:00:00 |
 35 |  97 | 2018-07-28 00:00:00 |
 87 |   8 | 2018-06-23 00:00:00 |
 71 |  41 | 2018-03-14 00:00:00 | 2018-03-13 00:00:00
 25 |  77 | 2018-09-19 00:00:00 |
 61 |  38 | 2018-08-07 00:00:00 |
(100 rows)
```

Relation Operatesat - DESCRIBE

```
palms=> \d operatesat
                        Table "public.operatesat"
   Column   |            Type             | Collation | Nullable | Default
------------+-----------------------------+-----------+----------+---------
 oid        | integer                     |           | not null |
 aid        | integer                     |           | not null |
 start_date | timestamp without time zone |           | not null | now()
 end_date   | timestamp without time zone |           |          |
Indexes:
    "operatesat_oid_aid_pk" PRIMARY KEY, btree (oid, aid)
Foreign-key constraints:
    "operatesat_aid_fk" FOREIGN KEY (aid) REFERENCES address(aid)
    "operatesat_oid_fk" FOREIGN KEY (oid) REFERENCES owner(oid)

palms=>
```

SELECT * FROM Operatesat:

```
 oid | aid |     start_date      |      end_date
-----+-----+---------------------+---------------------
  42 |  65 | 2018-06-11 00:00:00 |
  23 |  55 | 2018-05-01 00:00:00 |
   7 |  98 | 2018-07-17 00:00:00 |
  64 |  37 | 2018-05-12 00:00:00 |
  25 |  86 | 2018-04-03 00:00:00 | 2018-03-14 00:00:00
  98 |  38 | 2018-09-11 00:00:00 | 2018-02-27 00:00:00
  53 |  22 | 2018-01-27 00:00:00 |
  57 |  80 | 2018-08-10 00:00:00 | 2018-01-26 00:00:00
  72 |  92 | 2018-09-04 00:00:00 |
```

```
28 |  24 | 2018-06-01 00:00:00 |
80 |  82 | 2018-01-15 00:00:00 |
55 |  79 | 2018-05-01 00:00:00 | 2017-11-01 00:00:00
65 |  20 | 2018-05-16 00:00:00 |
29 |  51 | 2018-05-30 00:00:00 | 2018-07-02 00:00:00
92 |  58 | 2017-10-27 00:00:00 |
32 |  44 | 2018-08-19 00:00:00 |
 3 |  23 | 2018-05-16 00:00:00 |
12 |  19 | 2018-07-27 00:00:00 | 2017-11-15 00:00:00
22 |  22 | 2017-11-24 00:00:00 |
45 |  70 | 2018-06-08 00:00:00 | 2018-04-07 00:00:00
53 |  17 | 2017-12-27 00:00:00 | 2018-05-15 00:00:00
46 |  70 | 2018-04-06 00:00:00 |
20 |  85 | 2018-06-22 00:00:00 |
60 |  15 | 2018-06-27 00:00:00 | 2018-02-01 00:00:00
25 |  43 | 2018-02-10 00:00:00 |
27 |  21 | 2018-10-05 00:00:00 | 2017-12-05 00:00:00
28 |  20 | 2017-10-31 00:00:00 |
50 |  80 | 2018-04-29 00:00:00 |
33 |  10 | 2018-06-03 00:00:00 |
99 |  21 | 2018-01-05 00:00:00 |
93 |  12 | 2018-01-02 00:00:00 |
43 |  36 | 2018-08-12 00:00:00 | 2017-12-04 00:00:00
77 |  88 | 2018-03-17 00:00:00 |
46 |  64 | 2018-08-14 00:00:00 |
44 |  60 | 2018-04-18 00:00:00 | 2018-10-21 00:00:00
95 |  41 | 2018-05-12 00:00:00 |
25 |  92 | 2018-04-25 00:00:00 | 2017-12-29 00:00:00
63 |  85 | 2018-09-26 00:00:00 | 2018-05-20 00:00:00
91 |  49 | 2018-03-04 00:00:00 |
44 |  17 | 2018-04-27 00:00:00 |
34 |  68 | 2018-09-16 00:00:00 | 2018-09-19 00:00:00
16 |  55 | 2018-07-19 00:00:00 | 2017-12-27 00:00:00
10 |  71 | 2017-11-15 00:00:00 |
54 |  42 | 2018-09-17 00:00:00 |
44 |  51 | 2017-10-27 00:00:00 |
49 |  51 | 2018-02-16 00:00:00 |
94 |  65 | 2018-07-11 00:00:00 |
40 |  20 | 2017-12-27 00:00:00 |
20 |  37 | 2018-04-26 00:00:00 |
 1 |   2 | 2018-06-23 00:00:00 |
32 |  96 | 2018-04-30 00:00:00 | 2018-08-20 00:00:00
88 |  54 | 2018-07-15 00:00:00 |
47 |  69 | 2018-04-11 00:00:00 | 2018-05-17 00:00:00
73 |   1 | 2018-07-31 00:00:00 |
82 |  60 | 2017-12-04 00:00:00 | 2017-12-23 00:00:00
75 |  88 | 2018-09-20 00:00:00 |
57 |  10 | 2018-04-07 00:00:00 |
65 |   5 | 2018-06-08 00:00:00 |
64 |  68 | 2018-08-14 00:00:00 |
20 |  52 | 2018-08-10 00:00:00 |
45 |  57 | 2018-08-31 00:00:00 |
81 |  72 | 2018-07-21 00:00:00 | 2018-10-04 00:00:00
 7 |  23 | 2018-08-04 00:00:00 | 2017-11-30 00:00:00
74 |  28 | 2018-01-13 00:00:00 |
 3 |  94 | 2018-08-26 00:00:00 |
83 |  40 | 2018-05-17 00:00:00 |
69 |  94 | 2018-04-17 00:00:00 | 2017-12-01 00:00:00
39 |  72 | 2017-12-21 00:00:00 | 2018-06-03 00:00:00
61 |  63 | 2017-10-29 00:00:00 | 2018-10-22 00:00:00
11 |  32 | 2018-05-14 00:00:00 | 2018-08-11 00:00:00
93 |  39 | 2018-07-25 00:00:00 |
 4 |  37 | 2018-02-11 00:00:00 | 2018-01-23 00:00:00
64 |  31 | 2018-06-05 00:00:00 |
26 |  37 | 2018-09-18 00:00:00 |
43 |  90 | 2017-12-30 00:00:00 |
17 |  24 | 2018-04-15 00:00:00 |
37 |  28 | 2018-01-13 00:00:00 |
```

```
  48 |  90 | 2018-05-25 00:00:00 |
  71 |  58 | 2017-11-23 00:00:00 | 2018-07-13 00:00:00
  27 |  97 | 2018-01-20 00:00:00 | 2017-11-03 00:00:00
  81 |  62 | 2018-04-18 00:00:00 |
  48 |  94 | 2017-11-20 00:00:00 |
   2 |  82 | 2018-03-20 00:00:00 |
  17 |  65 | 2017-10-28 00:00:00 |
  62 |  26 | 2018-09-23 00:00:00 |
  82 |  69 | 2018-01-16 00:00:00 |
  10 |  86 | 2018-08-18 00:00:00 |
  16 |  38 | 2018-02-13 00:00:00 |
  69 |  89 | 2018-07-06 00:00:00 |
  92 |  75 | 2017-12-09 00:00:00 | 2018-02-15 00:00:00
   1 |  37 | 2018-08-30 00:00:00 |
  25 |  44 | 2018-01-17 00:00:00 |
  28 |  18 | 2018-08-25 00:00:00 |
  44 |  34 | 2018-09-30 00:00:00 |
  27 |  83 | 2018-09-22 00:00:00 |
  79 |   9 | 2018-07-01 00:00:00 |
   7 |  48 | 2018-06-03 00:00:00 |
  50 |  12 | 2018-06-20 00:00:00 | 2018-02-22 00:00:00
  23 |  40 | 2018-08-17 00:00:00 |
  73 |  81 | 2018-08-12 00:00:00 | 2018-09-14 00:00:00
(100 rows)
```

3.5 PostgreSQL Queries

Query 1 - List all properties with at least 2 bookings in San Francisco with prices more than $1000

The query:

```
SELECT
    p.*, count(b.bid) AS booking_count, a.city
FROM
    property p
INNER JOIN
    selects s ON s.pid = p.pid
INNER JOIN
    booking b ON b.pid = s.bid
INNER JOIN
    includes i ON i.pid = p.pid
INNER JOIN
    address a ON a.aid = i.aid
WHERE
    a.city = 'San Francisco'
GROUP BY
    p.pid, a.city
HAVING
    count(b.bid) >= 2
ORDER BY
    p.pid;
```

Output

```
 pid |    description    | rooms | size | price  | booking_count |     city
-----+-------------------+-------+------+--------+---------------+---------------
  16 | morbi quis tortor |       | 2462 | 184.92 |             2 | San Francisco
```

```
  34 | tristique est et  |     7 | 2725 |  66.09 |            2 | San Francisco
(2 rows)
```

Query 2 - List the top three guests in terms of spending, with their length of stay, the total cost of their most expensive trip. Additionally, only consider bookings for properties with at least two unique attractions

The query:

```
SELECT
    g.*,
    (b.cout::date -  b.cin::date) AS length_of_stay,
    b.daily_price,
    (b.cout::date -  b.cin::date) * b.daily_price AS total
FROM
    property p
INNER JOIN
    selects s ON s.pid = p.pid
INNER JOIN
    booking b ON b.pid = s.bid
INNER JOIN
    guest g ON g.gid = b.gid
WHERE
    b.daily_price > 150 AND
    EXISTS (
        SELECT *
        FROM attraction a1
        WHERE a1.pid = p.pid AND
        EXISTS (
            SELECT *
            FROM attraction a2
            WHERE a2.pid = p.pid AND
            a1.NAME != a2.NAME
        )
    )
ORDER BY
    total DESC
LIMIT 3;
```

Output

```
 gid | firstname | middlename |   lastname   |           email           | length_of_stay | daily_price |
 ↪  total
-----+-----------+------------+--------------+---------------------------+----------------+-------------+--------
  17 | Jesus     | Bordie     | Sidnell      | bsidnellg@newyorker.com   |              9 |       173.6 |
  ↪  1562.4
  89 | Karee     | Barry      | O' Mulderrig | bomulderrig2g@example.com |              3 |      316.97 |
  ↪  950.91
  52 | Uta       |            | Norkutt      | dnorkutt1f@phpbb.com      |              2 |      174.13 |
  ↪  348.26
(3 rows)
```

Query 3 - List guests who have at least one booking on each of the properties which appears in each of every city

The query:

```
SELECT
    g.*
FROM
    guest g
WHERE
    NOT EXISTS (
        SELECT
            *
        FROM
            property p
        WHERE
            NOT EXISTS (
                SELECT
                    *
                FROM
                    booking b
                INNER JOIN
                    selects s ON s.bid = b.bid
                WHERE
                    b.gid = b.gid
                    AND
                    b.pid = p.pid
            )
    );
```

Output

```
 gid | firstname | middlename | lastname | email
-----+-----------+------------+----------+-------
(0 rows)
```

Query 4 - List all properties with at least one booking with more than 3 views and with a price of more than $200

The query:

```
SELECT
    p.*
FROM
    property p
INNER JOIN
    booking b ON p.pid = b.pid
INNER JOIN
    VIEW v ON v.pid = p.pid
WHERE
    price > 200
GROUP BY
    p.pid, v.pid
HAVING
    count(v.pid) > 3
ORDER BY
    p.pid;
```

Output

```
 pid |       description        | rooms | size | price
-----+--------------------------+-------+------+--------
   3 | ante ipsum               |     4 |  963 | 244.59
  10 | volutpat convallis morbi |       | 2086 | 348.61
  12 | nisl nunc nisl           |     9 | 1182 | 227.72
  19 | in est                   |     5 | 1800 | 244.87
  21 | odio consequat varius    |     5 | 2988 | 292.94
```

Query 5 - List all owners that own two or more properties that are over 300 sq.ft., have more than 1 room, and offer a nature view

The query:

```
SELECT
    DISTINCT
    o.*
FROM
    OWNER o
INNER JOIN
    offers OF ON OF.oid = o.oid
INNER JOIN
    property p ON p.pid = OF.pid
WHERE
    p.size >= 300 AND
    p.rooms >= 1 AND
    EXISTS (
        SELECT *
        FROM VIEW v
        WHERE v.pid = p.pid
        AND v.NAME = 'Nature'
    )
GROUP BY
    o.oid
HAVING
    count(p.pid) >= 2
ORDER BY
    o.oid;
```

Output

```
oid | firstname  | middlename | lastname |         email
-----+------------+------------+----------+-----------------------
  49 | Pierette   | Davidde    | Culshaw  | dculshaw1c@example.com
  82 | Ermentrude | Jud        | Schankel | jschankel29@chron.com
(2 rows)
```

Query 6 - List guests that have never booked a property in Italy but give property rating of 4 or higher on average

The query:

```
SELECT
    DISTINCT
    g.*
FROM
    guest g
```

```
INNER JOIN
    booking b ON b.gid = g.gid
INNER JOIN
    selects s ON s.bid = b.bid
INNER JOIN
    property p ON p.pid = s.pid
LEFT OUTER JOIN -- null if address not included
    includes i ON i.pid = p.pid
LEFT OUTER JOIN -- null if address not included
    address a ON a.aid = i.aid
WHERE
    a.country != 'Italy' AND
    EXISTS (
        SELECT
            pr.gid
        FROM
            property_rating pr
        WHERE
            pr.gid = g.gid
        GROUP BY
            pr.gid, pr.rating
        HAVING
            avg(pr.rating) >= 4
    )
ORDER BY
    g.gid;
```

Output

```
 gid | firstname | middlename |  lastname  |              email
-----+-----------+------------+------------+---------------------------------
   9 | Shawn     | Wilone     | Pape       | wpape8@hp.com
  18 | Isobel    | Antoine    | Bodicum    | abodicumh@skyrock.com
  30 | Amby      | Miltie     | Lambden    | mlambdent@google.com
  43 | Eddi      | Sheri      | Mounch     | smounch16@google.ru
  50 | Adella    | Thornton   | Redmond    | tredmond1d@shareasale.com
  56 | Vittoria  | Rodolfo    | Ajsik      | rajsik1j@virginia.edu
  73 | Brynn     |            | Penella    | lpenella20@webs.com
  75 | Ken       | Julianna   | Crosscombe | jcrosscombe22@printfriendly.com
  76 | John      |            | Smith      | j.smith.coolio@arizona.edu
  86 | Felipe    |            | Gonnel     | kgonnel2d@uiuc.edu
  87 | Emile     | Erica      | Dayes      | edayes2e@dell.com
  88 | Oralee    | Eldridge   | Scade      | escade2f@home.pl
  91 | Wallas    | Quentin    | Brou       | qbrou2i@chron.com
  97 | Vance     |            | Whitlaw    | kwhitlaw2o@joomla.org
(14 rows)
```

Query 7 - List all properties with at least 4-star rating with price > 1000 (Excellent rated properties)

The query:

```
SELECT
    DISTINCT
    p.*
FROM
    property p
INNER JOIN
```

```
    property_rating pr ON pr.pid = p.pid
WHERE
    pr.rating >= 4
    AND
    p.price > 1000
ORDER BY
    p.pid;
```

Output

```
 pid |        description         | rooms | size |  price
-----+----------------------------+-------+------+---------
  76 | sodales scelerisque mauris |     8 | 2010 | 1315.80
(1 row)
```

Query 8 - List guests who have booked in Elm Street, the same street during Jane Doe's visit

The query:

```
SELECT
    DISTINCT
    g.*, b.cin, b.cout
FROM
    guest g
INNER JOIN
    booking b ON b.gid = g.gid
INNER JOIN
    selects s ON s.bid = b.bid
INNER JOIN
    property p ON p.pid = s.pid
WHERE p.pid IN (
    SELECT
        p2.pid
    FROM
        guest g2
    INNER JOIN
        booking b2 ON b2.gid = g2.gid
    INNER JOIN
        selects s2 ON s2.bid = b2.bid
    INNER JOIN
        property p2 ON p2.pid = s2.pid
    INNER JOIN
        includes i2 ON i2.pid = p2.pid
    INNER JOIN
        address a2 ON a2.aid = i2.aid
    WHERE
        g2.firstname = 'Jane' AND
        g2.lastname = 'Doe' AND
        a2.street = 'Elm Street' AND
        g.gid != g2.gid AND
        -- Dates overlap if
        -- other guest booked in before john checked out
        b.cin <= b2.cout AND
        -- and john checked in before other guest checked out
```

```
        b2.cin <= b.cout
)
ORDER BY
    g.gid;
```

Output

```
 gid | firstname | middlename | lastname | email | cin | cout
-----+-----------+------------+----------+-------+-----+------
(0 rows)
```

Query 9 - List guests who have booked the same property as John Smith's stay between 01/01/2017 - 01/05/2017

The query:

```
SELECT
    DISTINCT
    g.*, b.cin, b.cout
FROM
    guest g
INNER JOIN
    booking b ON b.gid = g.gid
INNER JOIN
    selects s ON s.bid = b.bid
INNER JOIN
    property p ON p.pid = s.pid
WHERE p.pid IN (
    SELECT
        p2.pid
    FROM
        guest g2
    INNER JOIN
        booking b2 ON b2.gid = g2.gid
    INNER JOIN
        selects s2 ON s2.bid = b2.bid
    INNER JOIN
        property p2 ON p2.pid = s2.pid
    INNER JOIN
        includes i2 ON i2.pid = p2.pid
    INNER JOIN
        address a2 ON a2.aid = i2.aid
    WHERE
        g2.firstname = 'John' AND
        g2.lastname = 'Smith' AND
        a2.street = 'Elm Street' AND
        g.gid != g2.gid AND
        -- Dates overlap if
        -- other guest booked in before john checked out
        b.cin <= b2.cout AND
        -- and john checked in before other guest checked out
        b2.cin <= b.cout
)
ORDER BY
    g.gid;
```

Output

```
 gid | firstname | middlename | lastname |       email        |         cin         |        cout
-----+-----------+------------+----------+--------------------+---------------------+---------------------
  29 | Wallache  | Merell     | Allett   | malletts@issuu.com | 2018-10-24 15:00:00 | 2018-10-29 12:00:00
  87 | Emile     | Erica      | Dayes    | edayes2e@dell.com  | 2018-10-23 15:00:00 | 2018-10-25 12:00:00
(2 rows)
```

Query 10 - List all available properties which have at least an ocean and a city view from 03/24/2019 to 04/12/2019 (Popular summer properties)

The query:

```
SELECT
    p.*
FROM
    property p
WHERE
    p.pid NOT IN (
        SELECT
            p2.pid
        FROM
            booking b2
        INNER JOIN
            selects s2 ON s2.bid = b2.bid
        INNER JOIN
            property p2 ON p2.pid = s2.pid
        WHERE
            b2.cin >= '03/24/2019' AND
            b2.cout <= '04/12/2019'
    )
    AND
    EXISTS (
            SELECT *
            FROM VIEW v1
            WHERE v1.pid = p.pid
            AND v1.NAME = 'Ocean'
    )
    AND
    EXISTS (
            SELECT *
            FROM VIEW v2
            WHERE v2.pid = p.pid
            AND v2.NAME = 'City'
    )
ORDER BY
    price, p.pid;
```

Output

```
 pid |      description       | rooms | size | price
-----+------------------------+-------+------+--------
  15 | dapibus dolor vel      |     2 | 1533 | 167.76
  11 | interdum venenatis     |     8 | 1252 | 264.63
   2 | molestie lorem quisque |     2 | 2556 | 267.41
(3 rows)
```

3.6 Data Loader

In this section we explain different variations of data loading techniques. We loaded an extensive amount of sample data into our physical implementation. We used different to accomplish this by writing script commands to insert data and using other software applications.

We loaded most of our data using a small SQL script by using \COPY. The use of \COPY isto load all rows in one command instead of using the INSERT commands. The \COPY command is optimized to be used when loading large numbers of rows, its much faster and gathers less overhead than in comparison to INSERT.

Moreover, this \COPY utility allows us to copy data into and out of tables in our database. It further suports modes such as:

- Binary mode
- TSV delimited (Tab Separated Values)
- CSV delimited (Comma Separated Values)

Below we can see this \COPY Postgres utility in action:

Example Usage of COPY PSQL Command

```
1 \COPY OWNER FROM 'owner.csv' WITH CSV NULL AS '' HEADER;
2
3 SELECT setval('owner_oid_seq',SELECT MAX(oid) FROM OWNER));
```

After the data is loaded into the table, in this case, owner table, the primary key sequence numbers are unfortunately out of sync. It becomes necessary then to reset the key sequence to highest value of the primary key in that table. This step is then repeated for every CSV file of each respective sample data, from phase 2, to complete the data loading procedure.

4 Phase 4: PostgreSQL Database Management System PL/pgSQL Components

Under Phase 3, Postgress SQL Database Management System, we discussed the creation of a logical and physical database with Postgress DBMS. We further detailed some operations and syntax used in PostgreSQL.

However, we must note creating a professional, coherent and efficient database requires more complexity. For this reason we will begin to analyze and study the components of Postgres GPL/SQL in this phase. In the next few sections, we define the purpose of PL/SQL then we extract information how its features and syntax. After getting a clear understanding of PL/SQL, we can implement some samples in our database and further explore some extensions offered by other DBMS.

4.1 Postgres PL/pgSQL

PL/pgSQL is PostgresSQL's loadable procedural language extension of SQL and it allows us and users to create a *flow control structures* that are quite related to loops and conditional statements. PL/pgSQL allows users to create stored procedures and create functions that can be used anywhere that it is allowed.

There are several advantages to using stored procedures and functions. Some of PL/pgSQL's built-in functions are quite powerful because it allows us to manage complex tasks. Otherwise, we would have to manually input and manipulate blocks of PL/pgSQL code. Below is an overview of how advantageous PL/pgSQL can be:

- Ease of use
- Possibly be defined to be trusted by the server
- Inherits all user-defined types, functions and operations
- Perform complex computations
- Adds control structures to SQL language
- Be used to create functions and trigger procedures

4.1.1 PL/pgSQL Program Structure, Control Statements, and Cursors

PL/pgSQL is a block-structured language and its program structure is created by groups of statements and declarations that are considered as a *block*.

PL/pgSQL Block

```
[ <<label>> ]
[ DECLARE
    declarations ]
BEGIN
    statements
END [ label ];
```

The PL/pgSQL block's declaration and statement within a block ends with a semicolon, while a block that appears inside another block has an END with a semicolon in the end. Moreover, all words in the block are case sensitive. A statement in a section of the statement can be considered a sublock. Sublocks are used for logical grouping in small groups of statements.

Syntax for conditional statements:

```
IF boolean-expression THEN
    statements
[ ELSIF boolean-expression THEN
    statements
[ ELSIF boolean-expression THEN
    statements
    ...]]
[ ELSE
    statements ]
END IF;
```

Syntax for basic loop

```
[ <<label>> ]
LOOP
    statements
END LOOP [ label ];
```

Syntax for while loop

```
[ <<label>> ]
WHILE boolean-expression LOOP
    statements
END LOOP [ label ];
```

Syntax for for loop

```
[ <<label>> ]
FOR name IN [ REVERSE ] expression .. expression [ BY expression ] LOOP
    statements
END LOOP [ label ];
```

A *cursor* allows us to encapsulate a query and further process one each row at a time. It is advised to use cursors when we would like to divide and conquer in order to maximamize performance. If we process large number of rows, we may run the risk of memory overflow. The SCROLL is for backwards fetches, while the NO SCROLL tells us that the any backward fetch will be rejected. But if neither appears then it tis query-dependent whether the backwards fetch will be allowed or not. Below the syntax for cursor procedures:

Syntax for Cursor Procedures

```
name [ [ NO ] SCROLL ] CURSOR [ ( arguments ) ] FOR query;
```

4.1.2 Stored Procedures

PostgreSQL is a procedural language, therefore it is referred to as *stored procedures*. The stored procedures add procedural elements such as control structures, loops, declarations, assignments, flow-of-control and others.

Moreover, the purpose of a stored procedures in most cases is to return one or more results, return one or more scalar values as OUT perameters. In PostgreSQL stored procedures usually do not return any value or return one or more result sets. However, if a stored Procedure does not return any value then we can simply specify VOID as a return type. We can return a result set from PostgreSQL procedure by using *refcursor* return type but the cursor must be closed immediately after the procedure call. The stored procedures are created with CREATE FUNCTION as shown:

Syntax for Stored Procedures:

```
CREATE [ OR REPLACE ] FUNCTION
    name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ] [, ...] ] )
    [ RETURNS rettype
      | RETURNS TABLE ( column_name column_type [, ...] ) ]
  { LANGUAGE lang_name
    | WINDOW
    | IMMUTABLE | STABLE | VOLATILE
    | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    | COST execution_cost
    | ROWS result_rows
    | SET configuration_parameter { TO value | = value | FROM CURRENT }
    | AS 'definition'
    | AS 'obj_file', 'link_symbol'
  } ...
    [ WITH ( attribute [, ...] ) ]
```

4.1.3 Stored Functions

When it comes to *User-Defined Functions (UDF)* is quite similar to *stored procedures* calls with the exception that it can not return multiple result sets and can not return values as OUT perameters. However, it is similar to stored procedures in that user-defined functions can be invoked any time, be created with CREATE FUNCTION statement, be used in an expression, return a value, and return a single result set as a table function. The objective of user-defined functions is to process the input parameters and return a new value.

Syntax of a User-Defined Function:

```
CREATE [ OR REPLACE ] FUNCTION
    name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ] [, ...] ] )
    [ RETURNS rettype
      | RETURNS TABLE ( column_name column_type [, ...] ) ]
  { LANGUAGE lang_name
    | WINDOW
    | IMMUTABLE | STABLE | VOLATILE
    | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    | COST execution_cost
    | ROWS result_rows
    | SET configuration_parameter { TO value | = value | FROM CURRENT }
    | AS 'definition'
    | AS 'obj_file', 'link_symbol'
  } ...
    [ WITH ( attribute [, ...] ) ]
```

4.1.4 Trigger Procedures

Trigger Procedures is stored PL/pgSQL procedure associated with with a specific table. Triggers are activating based on either update, delete, or insertion to a table. A trigger is created in two parts, first a function must be created with the CREATE FUNCTION command. It is quite similar to the other functions however it does not take any arguments a return type of *trigger*. The function must be declared without any arguments even if it expects to receive arguments that are specified. Then, a CREATE TRIGGER command is used to attach the function to a table as a trigger.

A trigger function has to return a NULL or row having the same structure of the the table the trigger was fired for. The trigger function provides two kinds of triggers which is *row level trigger* (FOR EACH ROW) and *statement level trigger* (FOR EACH STATEMENT). The OLD and NEW represents the states of a row in the table before or after the trigger occurs.

Syntax of a Trigger Procedure:

```
CREATE [ CONSTRAINT ] TRIGGER name { BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }
    ON table_name
    [ FROM referenced_table_name ]
    [ NOT DEFERRABLE | [ DEFERRABLE ] [ INITIALLY IMMEDIATE | INITIALLY DEFERRED ] ]
    [ REFERENCING { { OLD | NEW } TABLE [ AS ] transition_relation_name } [ ... ] ]
    [ FOR [ EACH ] { ROW | STATEMENT } ]
    [ WHEN ( condition ) ]
    EXECUTE PROCEDURE function_name ( arguments )
```

4.2 Postgres PL/pgSQL Subprograms

In this section, we focus on on Postgres PL/pgSQL subprograms which encompass concentration on implementing procedures, functions and triggers. We will illustrate three-stored procedures and three-triggered procedures.

Postgres Function 1: Insert Function - Make a Booking

Description:

This postgres function will validate that properties were selected and were available for the requested check in and check out dates. Additionally, it will calculate the total sum of the price for all the properties that are selected using aggregated sum of the properties' price, and store it in daily_price of the booking relation that will be created. Lastly, it will insert the property selections into the table 'selects' to mark those properties as a part of the booking.

```
1  CREATE OR REPLACE FUNCTION make_booking(
2      IN guest_id integer,
3      IN requested_checkin timestamp without time zone,
4      IN requested_checkout timestamp without time zone,
5      IN requested_property_ids integer[]
6  )
7      RETURNS integer -- booking id, if successful
8      LANGUAGE plpgsql
9  AS
10 $$
11 DECLARE
12     check_property_id integer;
13     daily_price_total decimal(16,2);
14     booking_id integer;
15     property_id integer;
16 BEGIN
17     -- Validate that at least one property was selected
18     IF requested_property_ids = '{}'::int[] THEN
19         RAISE EXCEPTION 'No properties were selected for booking';
20     END IF;
21
```

```
    -- Validate that the selected properties are available
    FOR check_property_id IN
        SELECT
            p.pid
        FROM
            booking b
        INNER JOIN
            selects s ON s.bid = b.bid
        INNER JOIN
            property p ON p.pid = s.pid
        WHERE
            b.cin >= requested_checkin AND
            b.cout <= requested_checkout AND
            p.pid = ANY(requested_property_ids) AND
            b.booking_status NOT IN ('Canceled', 'Void')
    LOOP
        RAISE EXCEPTION 'Property pid: % is not available', check_property_id;
    END LOOP;
    -- Calculate the total price for all selected properties
    SELECT
        SUM(p.price) INTO daily_price_total
    FROM
        property p
    WHERE
        p.pid = ANY(requested_property_ids)
    GROUP
        BY p.pid;
    BEGIN
        INSERT
            INTO
                booking (booking_date, daily_price, cin, cout, gid, booking_status)
            VALUES
                (CURRENT_TIMESTAMP, daily_price_total, requested_checkin,
                ↪  requested_checkout, guest_id, 'Pending')
            RETURNING
                bid INTO booking_id;
        FOREACH property_id IN ARRAY requested_property_ids
        LOOP
            INSERT
                INTO
                    selects (bid, pid)
                VALUES
                    (booking_id, property_id);
        END LOOP;
    END;
    RETURN booking_id;
END;
$$;
```

Postgres Function 1 - Usage:

```
-- 1) Must select a property error
SELECT * FROM make_booking(53, '11/19/2018', '11/25/2018', ARRAY[]::int[]);
```

```
3
4  -- 2) Succeeds
5  SELECT * FROM make_booking(53, '11/19/2018', '11/25/2018', ARRAY[1,2]);
6
7  -- 3) This time, property 2 is already booked, therefore error
8  SELECT * FROM make_booking(53, '11/19/2018', '11/25/2018', ARRAY[2,3]);
```

Postgres Function 1 - Output:

```
1  -- 1)
2
3  ERROR:  No properties were selected for booking
4  CONTEXT:  PL/pgSQL function make_booking(integer,timestamp without time zone,timestamp
   ↪  without time zone,integer[]) line 10 at RAIS
5
6  -- 2)
7
8   make_booking
9  --------------
10           101
11 (1 row)
12
13 -- 3)
14
15 ERROR:  Property pid: 2 is not available
16 CONTEXT:  PL/pgSQL function make_booking(integer,timestamp without time zone,timestamp
   ↪  without time zone,integer[]) line 29 at RAISE
```

Postgres Function 2: Delete function - Delete a property

Description:

This function will delete a property if possible to do so. The function will clean up any addresses and those that are no longer referenced by any other table. Additionally, property_ratings, amenity, view, attraction are cleared for any records relating to that property.

```
1  -- Validate and process delete on orders
2  CREATE OR REPLACE FUNCTION delete_property(
3      IN property_id integer
4  )
5      RETURNS integer
6      LANGUAGE plpgsql
7  AS
8  $$
9  BEGIN
10     DECLARE
11         booking_id integer;
12         guest_id integer;
13         address_id integer;
14     BEGIN
15         -- Validate that we are allowed to delete a property
16         FOR booking_id, guest_id IN (
17             SELECT
18                 b.bid, b.gid
19             FROM
20                 booking b
21             INNER JOIN
22                 selects s ON s.bid = b.bid
23             INNER JOIN
24                 property p ON p.pid = s.pid
25             WHERE
26                 -- ignore void bookings, they can be deleted
27                 b.booking_status != 'Void' AND
28                 p.pid = property_id
29             )
30         LOOP
31             RAISE EXCEPTION 'Property pid:% already has a permanent booking record bid: %
               ↪  by guest gid:%', property_id, booking_id, guest_id;
32         END LOOP;
33
34         FOR address_id IN (
35             -- Delete all included address for property
36             -- and retain the list of address IDs
37             WITH address_id_table AS (
38                 DELETE
39                 FROM
40                     includes i
41                 WHERE
42                     i.pid = property_id
43                 RETURNING
```

```
                    i.aid
            )
            SELECT * FROM address_id_table
        )
        LOOP
            -- Then delete all addresses that are
            -- not used by any other foreign keys
            DELETE FROM
                address a
            WHERE
                NOT EXISTS (
                    SELECT la.aid FROM livesat la WHERE la.aid = a.aid
                ) AND NOT EXISTS(
                    SELECT oa.aid FROM operatesat oa WHERE oa.aid = a.aid
                );
        END LOOP;
        -- purge property ratings, amenities, views, attractions for property
        DELETE FROM property_rating pr WHERE pr.pid = property_id;
        DELETE FROM amenity am WHERE am.pid = property_id;
        DELETE FROM VIEW vi WHERE vi.pid = property_id;
        DELETE FROM attraction AT WHERE AT.pid = property_id;
        -- Lastly, the property itself
        DELETE FROM property p    WHERE p.pid = property_id;

        RETURN property_id;
    END;
END;
$$;
```

Postgres Function 2 - Usage:

```
-- Property pid 1 already has a booking pending created in procedure 1
 SELECT * FROM delete_property(1);

 -- Will succeed to delete, as no booking exists for property 3
SELECT * FROM delete_property(3);
```

Postgres Function 2 - Output:

```
----------------------------------------------------------------------
-- 1)

ERROR:  update or delete on table "property" violates foreign key constraint
↪  "offers_pid_fk" on table "offers"
DETAIL:  Key (pid)=(1) is still referenced from table "offers".
CONTEXT:  SQL statement "DELETE FROM property p         WHERE p.pid = property_id"
PL/pgSQL function delete_property(integer) line 59 at SQL statement

-- 2)

 delete_property
```

```
12 -----------------
13                3
14 (1 row)
```

Postgres Function 3: Average Property Sales By Month/Year

Description:

Calculates the average property sales for every month of the year for which booking records exist. The property id is optional in this case. The first step is to calculate the number of days the booking is for, the difference of check out and check in dates. Secondly, it calculates the total price of the stay. Lastly, the aggregate function average is used to calculate the average price.

```
CREATE OR REPLACE FUNCTION avg_property_sales(IN prop_id int)
    RETURNS TABLE (
        pid int,
        date date,
        amount numeric
    )
LANGUAGE plpgsql
AS
$$
BEGIN
    RETURN QUERY
        SELECT
            p.pid,
            date_trunc('month', b.booking_date)::date AS year_month,
            avg(
                (b.cout::date - b.cin::date) * b.daily_price
            ) AS amount
        FROM
            booking b
        INNER JOIN
            selects s ON s.bid = b.bid
        INNER JOIN
            property p ON p.pid = s.pid
        WHERE
            b.booking_status = 'Completed'
        GROUP BY
            p.pid, year_month
        HAVING
            prop_id IS NULL OR p.pid = prop_id
        ORDER BY
            year_month;
END
$$;
```

Postgres Function 3 - Usage:

```
-- 1) Get the average property sales for all months of property pid = 89
SELECT * FROM avg_property_sales(89);

```

```
-- 2) Equivalent to the above, using a query on the result:
SELECT * FROM avg_property_sales(NULL) WHERE pid = 89;

-- 3) Find the highest average sale month of each property, top 5
SELECT * FROM avg_property_sales(NULL) ORDER BY amount DESC LIMIT 5;
```

Postgres Function 3 - Output:

```
-- 1)

 pid |    date    |        amount
-----+------------+-----------------------
  89 | 2018-03-01 | 2656.7100000000000000
(1 row)

-- 2)

 pid |    date    |        amount
-----+------------+-----------------------
  89 | 2018-03-01 | 2656.7100000000000000
(1 row)

-- 3)

 pid |    date    |         amount
-----+------------+------------------------
   5 | 2018-01-01 | 10625.4400000000000000
  30 | 2018-08-01 |  2829.4400000000000000
  53 | 2018-08-01 |  2829.4400000000000000
  89 | 2018-03-01 |  2656.7100000000000000
  71 | 2018-04-01 |  1497.6000000000000000
(5 rows)
```

Postgres Trigger 1: Before Update Trigger - Property Id Before

Description:

In the event that a property id is changed, the change must be propagated throughout the many relations. This trigger will update the selects, amenity, attraction, view, property_rating, offers, and includes relation with the new replacement primary key of property.

```
CREATE OR REPLACE FUNCTION update_property_id()
    RETURNS TRIGGER
    LANGUAGE plpgsql
AS
$$
BEGIN
    -- Defer constraints until end of transaction
    SET CONSTRAINTS
        selects_pid_fk,
        amenity_pid_fk,
        attraction_pid_fk,
        view_pid_fk,
        property_rating_pid_fk,
        offers_pid_fk,
        includes_pid_fk
        DEFERRED;
    UPDATE selects s    SET pid = NEW.pid WHERE pid = OLD.pid;
    UPDATE amenity        SET pid = NEW.pid WHERE pid = OLD.pid;
    UPDATE attraction    SET pid = NEW.pid WHERE pid = OLD.pid;
    UPDATE VIEW        SET pid = NEW.pid WHERE pid = OLD.pid;
    UPDATE property_rating    SET pid = NEW.pid WHERE pid = OLD.pid;
    UPDATE offers        SET pid = NEW.pid WHERE pid = OLD.pid;
    UPDATE includes        SET pid = NEW.pid WHERE pid = OLD.pid;
    RETURN NEW;
END;
$$;

DROP TRIGGER IF EXISTS update_property_id_trigger ON property;
CREATE TRIGGER update_property_id_trigger
    BEFORE UPDATE ON property
    FOR EACH ROW EXECUTE PROCEDURE update_property_id();
```

Postgres Trigger 1 - Usage:

```
-- 1) Change property id from 5 --> 999
UPDATE property
SET pid = 999
WHERE pid = 5;
```

Postgres Trigger 1 - Output:

```
-- 1)
UPDATE 1
```

Postgres Trigger 2: Cascade Deletion Trigger - Cascade Delete Booking

Description:

A booking may not be delete after is has been completed. However, before it get to that that stage, it is possible and safe to delete it as long as there is no reference integrity violations are found, then they are deleted. There is a simply cascade deleted.

```
CREATE OR REPLACE FUNCTION delete_booking()
    RETURNS TRIGGER
    LANGUAGE plpgsql
AS
$$
BEGIN
    IF OLD.booking_status <> 'Completed' THEN
        SET CONSTRAINTS property_rating_bid_fk DEFERRED;
        DELETE FROM property_rating WHERE bid = OLD.bid;
        DELETE FROM selects WHERE bid = OLD.bid;
        RETURN OLD;
    ELSE
        -- No completed bookings may be deleted
        RAISE EXCEPTION 'Booking bid: % has already been completed', OLD.bid;
    END IF;
END;
$$;

DROP TRIGGER IF EXISTS delete_booking_trigger ON booking;
CREATE TRIGGER delete_booking_trigger
    BEFORE DELETE ON booking
    FOR EACH ROW EXECUTE PROCEDURE delete_booking();
```

Postgres Trigger 2 - Usage:

```
-- 1) Attempt to delete a completed booking
DELETE FROM booking
WHERE bid = 5;

-- 2) delete a booking that was cancelled (legal)
DELETE FROM booking WHERE bid = 1;
```

Postgres Trigger 2 - Output:

```
-- 1) As expected, unable to delete completed bookings

ERROR:  Booking bid:5 has already been completed
CONTEXT:  PL/pgSQL function delete_booking() line 8 at RAISE

-- 2) Booking is then deleted as a result
```

```
7
8 DELETE 1
```

Postgres Trigger 3: Instead Of Trigger - Instead Of View Update

Description:

The first step to this trigger is to create a view 'booking_property_info' composed of mainly booking and property tables. The trigger will update each associated table individually, tables property, and booking, and in the event of property not being selected, it will insert the selection into 'selects' table.

```
CREATE OR REPLACE VIEW booking_property_info AS
    SELECT DISTINCT
        b.*, p.*
    FROM
        booking b
    LEFT OUTER JOIN
        selects s ON s.bid = b.bid
    LEFT OUTER JOIN
        property p ON p.pid = s.pid
    ORDER BY
        p.pid;

CREATE OR REPLACE FUNCTION update_booking_property_info()
    RETURNS TRIGGER
    LANGUAGE plpgsql
AS
$$
BEGIN
    UPDATE
        booking
    SET
        daily_price = NEW.daily_price,
        booking_date = NEW.booking_date,
        cin = NEW.cin,
        cout = NEW.cout,
        booking_status = NEW.booking_status,
        confirm_nr = NEW.confirm_nr,
        gid = NEW.gid
    WHERE
        bid = OLD.bid;

    UPDATE
        selects
    SET
        pid = NEW.pid
    WHERE
        bid = OLD.bid AND
        pid = OLD.pid;

    IF NOT EXISTS (
        SELECT *
        FROM selects s
```

```
        WHERE s.bid = OLD.bid AND s.pid = NEW.pid
    ) THEN
        INSERT INTO selects (bid, pid) VALUES (OLD.bid, NEW.pid);
    ELSE
        UPDATE
            property
        SET
            description = NEW.description,
            rooms = NEW.rooms,
            size = NEW.size,
            price = NEW.price
        WHERE
            pid = OLD.pid;
    END IF;
    RETURN NEW;
END;
$$;

DROP TRIGGER IF EXISTS update_booking_property_info_trigger
    ON booking_property_info;
CREATE TRIGGER update_booking_property_info_trigger
    INSTEAD OF UPDATE ON booking_property_info
    FOR EACH ROW EXECUTE PROCEDURE update_booking_property_info();
```

Postgres Trigger 3 - Usage:

```
-- 1) Check the current value of booking 5 in booking_property_info

SELECT bid, booking_status, pid FROM booking_property_info WHERE bid = 5;

-- 2) Update row in booking_property_info

UPDATE booking_property_info
SET booking_status = 'Canceled', pid = 1
WHERE bid = 5;

-- 2) Validate that the row has now changed

SELECT bid, booking_status, pid FROM booking_property_info WHERE bid = 5;
```

Postgres Trigger 3 - Output:

```
-- 1) First verify the actual datain the database

bid | booking_status | pid
-----+----------------+-----
   5 | Canceled       |  92
(1 row)


-- 2) Update result, 1 row in view updated
```

```
10
11 UPDATE 1
12
13 -- 2) As expected, booking is now completed for property 1
14
15  bid | booking_status | pid
16 -----+----------------+-----
17    5 | Completed      |   1
18 (1 row)
```

4.3 PL/pgSQLLike Tools in Microsoft SQL Server and PostgreSQL DBMS

The physical database implementation for Palms Vacation Properties is based on PostgreSQL's PL/pgSQL. The aforementioned is one of many commercial grade database management systems software available in the market for users and designers alike. In this section, we focus on other tools that offer related functionalities that are similar and uniquely different accordinly.

We will be investigating Microsoft Server's (Transact-SQL), MySQL's and Postgres PL/pgSQL's ways of writing procedures. We are reporting on the similarities and differences of creating stored procedures and stored functions among the three DBMS (MS SQL Server, MySQL and Postgres PL/pgQL). Additionally, we will explore their syntax of selective-statements, repetitive-statements, subprogram structures, perameter passing methods.

Microsoft SQL Server: Transact-SQL
Comparison of T-SQL to other commercial DBMS languages

The *Transact-SQL* also known as simply *T-SQL* is a proprietary procedural language used by Microsoft in SQL server. It is Microsoft's and Sybase's proprietary extension of the SQL used to interact with relational databases but expanded to include various characteristics such as procedural programming, local variable, support functions for string processing among others.

T-SQL adds a number of features that are otherwise not in other languages. The T-SQL provides set of local variable include DECLARE, SET and SELECT. Some key words for flow control include BREAK (ends enclosing WHILE loop), BEGIN and END (for block of statements), CONTINUE (for next interation of loop to execute), GOTO, IF-ELSE (condition execution), RETURN, WAITFOR (used for waiting a specific time), and WHILE.

Not only that but also T-SQL offers encryption methods with stored procedures. T-SQL offers TRY CATCH logic to support exception handling, this helps users simplify code and saves some time after each SQL exection statement. The TRY CATCH procedure is different in T-SQL because unlike its competitors specifically MySQL, it offers exception handling in

different sections.

However, T-SQL does not allow procedures to be grouped in packages. Also, the syntax for passing a perameter in T-SQL is that of an @ character which always begins any perameters. The syntax difference is unique to T-SQL and does not apply to MySQL and PL/pgSQL. T-SQL does not offer *for* loops, it just offers basic *while* loops.

Syntax for creating a stored procedure/function

```
CREATE [ OR ALTER ] { PROC | PROCEDURE }
    [schema_name.] procedure_name [ ; number ]
    [ { @parameter [ type_schema_name. ] data_type }
        [ VARYING ] [ = default ] [ OUT | OUTPUT | [READONLY]
    ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
[ FOR REPLICATION ]
AS { [ BEGIN ] sql_statement [;] [ ...n ] [ END ] }
[;]

<procedure_option> ::=
    [ ENCRYPTION ]
    [ RECOMPILE ]
    [ EXECUTE AS Clause ]
```

Syntax for basic loop

```
WHILE Boolean_expression
     { sql_statement | statement_block | BREAK | CONTINUE }
```

MySQL

Comparison of MySQL to other commercial DBMS languages

MySQL is an open-source and free relational database management system that is now owned by Oracle Corporation. MySQL is offered under two editions: The open source MySQL Community Server and the propietary Enterprise Server. The propietary edition offers additional paid editions that includes more functionality. MySQL is written in C and C++, and works in many system platforms. MySQL is quite popular among large-scale web-based applications such as YouTube, Google, Twitter and others.

MySQL is quite similar its competitors T-SQL and PostgreSQL. It offers stored procedures, triggers, cursors, updatable views, query caching, full-text indexing and text searching among others. However, MySQL does not offer *for* loops, only basic loops are supported but offers control flow statements. Also, it does not offer packages for namespace management similar T-SQL.

However, parameters are passed in the same manner as PL/pgSQL and do not use @ character like T-SQL. One key difference that separates MySQL among the other DBMS

languages is the way it parses its stored procedures. It uses a *delimeter* command to make a change, the default end-line character form a semicolon to a //. It is also limited in areas of fault tolerance and performance diagnostics. It suffers from poor performance scaling and its functionality is dependent on add-ons and applications such as text search.

Syntax for creating a procedure/function:

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body

CREATE
    [DEFINER = { user | CURRENT_USER }]
    FUNCTION sp_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body

proc_parameter:
    [ IN | OUT | INOUT ] param_name type

func_parameter:
    param_name type

type:
    Any valid MySQL data type

characteristic:
    COMMENT 'string'
  | LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }

routine_body:
    Valid SQL routine statement
```

Syntax for a simple loop:

```
[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

PostgreSQL: PL/pgSQL
Comparison of PL/SQL to other commercial DBMS languages

PL/pgSQL is a procedural programming language that is supported by Postgres, it is quite similar to Oracle Corporation's PL/SQL language. It was influenced by PL/SQL and

Ada. PL/pgSQL was designed by Jan Wieck and written in C. PL/pgSQL was used to implement the physical database for Palms Vacation Properties.

PL/pgSQL is quite popular and used for web databases. PL/pgSQL is not propietary, its open-source and free like some parts of MySQL. Some users claim that it has better support than the propietary vendors when compared to T-SQL and MySQL. It is known to be reliable and quite stable. It allows users to manage structured and unstructured data. The database management engine is scalable and it can handle large data, supports JSON, offers a wide variety of predefined functions, and a number of interfaces are available. PL/PgSQL has support for conditional statements, basic loops, *for while* loops, *for* loops among others. There are many advantages to PL/pgSQL such as what we aforementioned in section 4.0.0:

- Ease of use
- Possibly be defined to be trusted by the server
- Inherits all user-defined types, functions and operations
- Perform complex computations
- Adds control structures to SQL language
- Be used to create functions and trigger procedures

One of the few disadvantages to PL/pgSQL is that it also does not support packages similar MySQL and T-SQL but instead it uses *schemas* to organize functions into groups. Since it does not support packages, it also does not support package-level variables either, but can keep per-session state in temporary tables. Also, it is noticeably slower in contrast to MySQL.

Syntax for creating a procedure/function:

```
CREATE [ OR REPLACE ] FUNCTION
    name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ] [, ...] ] )
    [ RETURNS rettype
      | RETURNS TABLE ( column_name column_type [, ...] ) ]
  { LANGUAGE lang_name
    | WINDOW
    | IMMUTABLE | STABLE | VOLATILE
    | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    | COST execution_cost
    | ROWS result_rows
    | SET configuration_parameter { TO value | = value | FROM CURRENT }
    | AS 'definition'
    | AS 'obj_file', 'link_symbol'
  } ...
    [ WITH ( attribute [, ...] ) ]
```

Syntax for while loops:

```
[ <<label>> ]
WHILE boolean-expression LOOP
    statements
END LOOP [ label ];
```

Syntax for for loops:

```
[ <<label>> ]
FOR name IN [ REVERSE ] expression .. expression [ BY expression ] LOOP
    statements
END LOOP [ label ];
```

5 Phase 5: Graphics User Interface Design and Implementation

In this phase, we showcase and implement a database application with a graphical user Interface (GUI). In the next few sections we will demonstrate segments of our database, test it and exhibit its applications. We provide itemized descriptions of the GUI application, display screenshots of the application, run an assessment of the tables, views of the stored subprograms, and triggers used. Then, we discuss the programming side and implementation process of our application. We achieve this by explaining and presenting specific components server-side programming, middle-tier programming, and client-side programming. Lastly, we include any relevant code snippets, and provide a survey of what was achieved and learned over the course of the development of Palms Vacation Properties database and web application.

5.1 Functionalities and User group of the GUI application

Before we can begin to design a front end application for Palms Vacation Properties database, we must acknowledge that it will be used by a vast number of users. Therefore, we must toggle and tailor the application to enclose all the differences and similaries of the users. However, this application focuses on the interactions of a customers *(guests)*.

5.1.1 Itemized Descriptions

The following is an itemized description of the possible users of the GUI application for Palms Vacation properties:

Guest Users
Guest users require the use of searching for properties, choose vacation dates, go through the booking process and get an invoice with booking information.

- View and choose location of properties
- View and choose travel dates (i.e check-in and check-out dates)
- View property prices
- View and filter amenities
- View and choose attractions

- View and filter through panormamic views
- Create a booking
- View property information
- View booking invoice

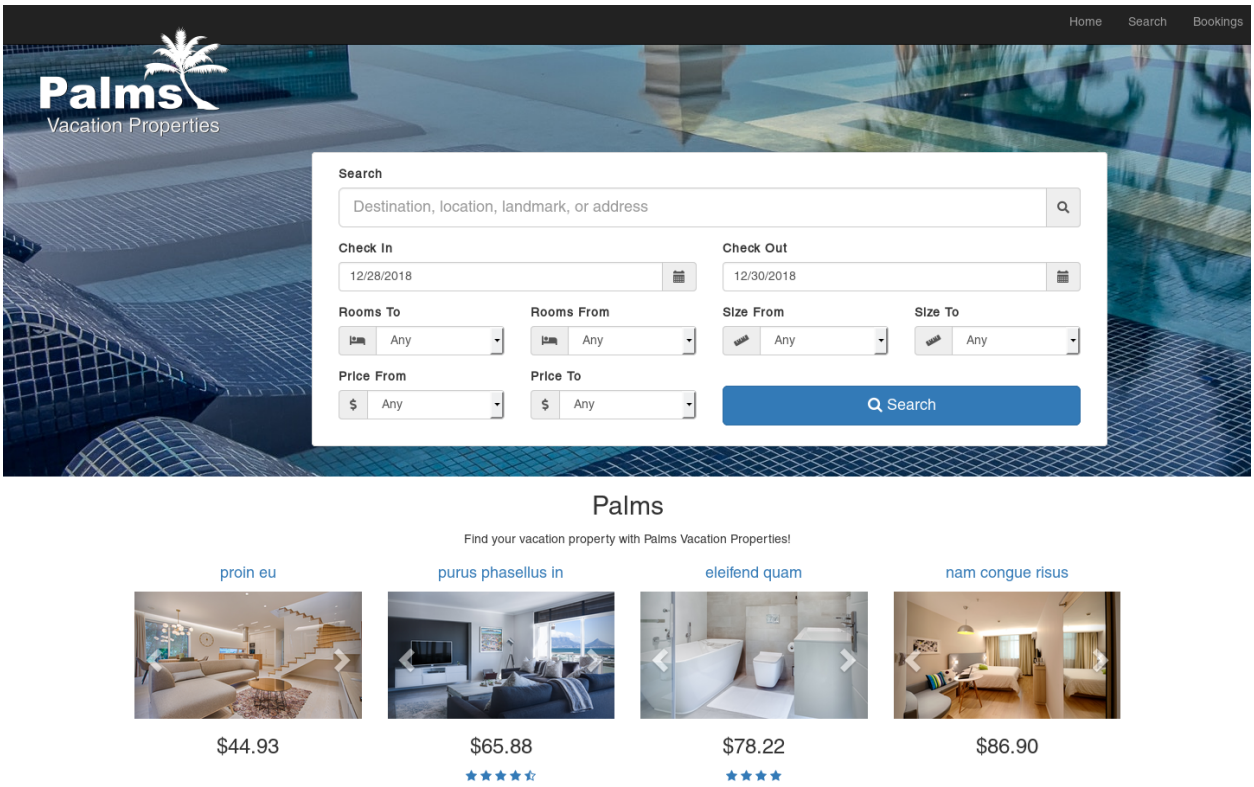
Owner Users

Owner users require the use of adding properties to the application. Also, they need to be able to manage the property among other things.

- Add a property
- Add type of property (e.g condo, room and house)
- Add property details (i.e address, amenities, descriptions and photos)

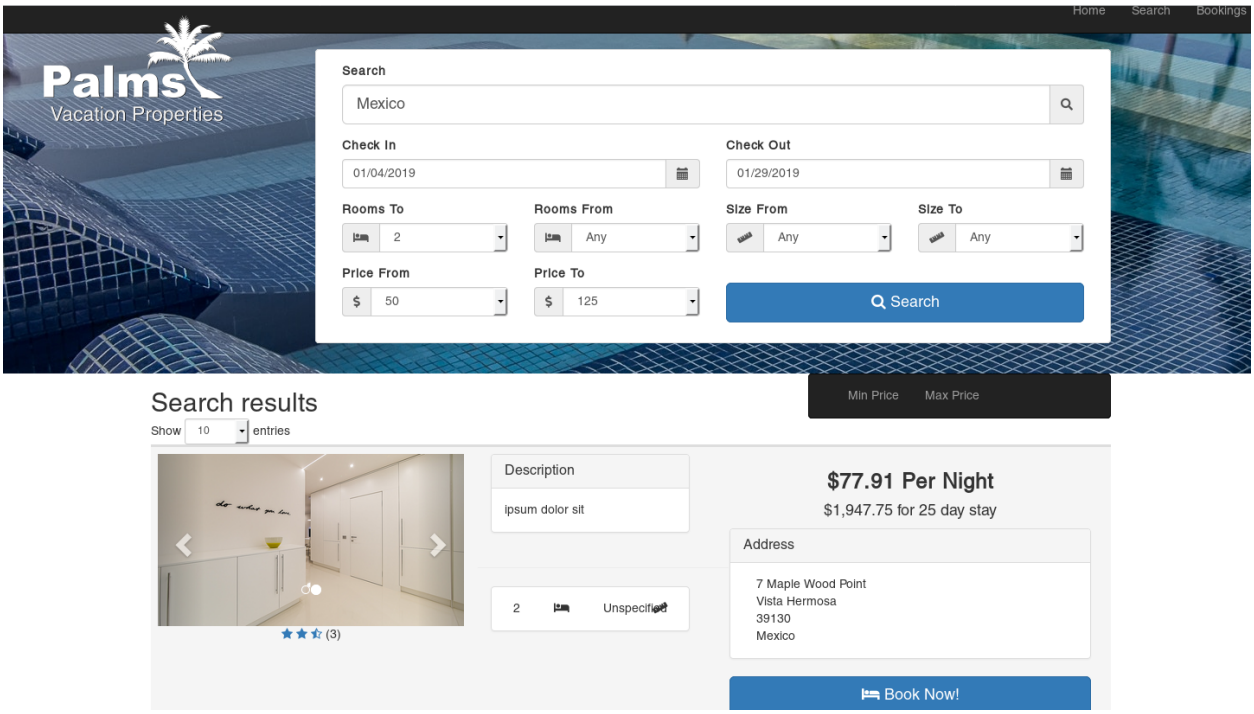
5.1.2 Screenshots of the Application

This section deals wih several subsections that screenshot the applications functionality. We describe how each functionality ties into the implementation of the databse. We give a description of the functionalities of the command button, contents of the data and so forth.



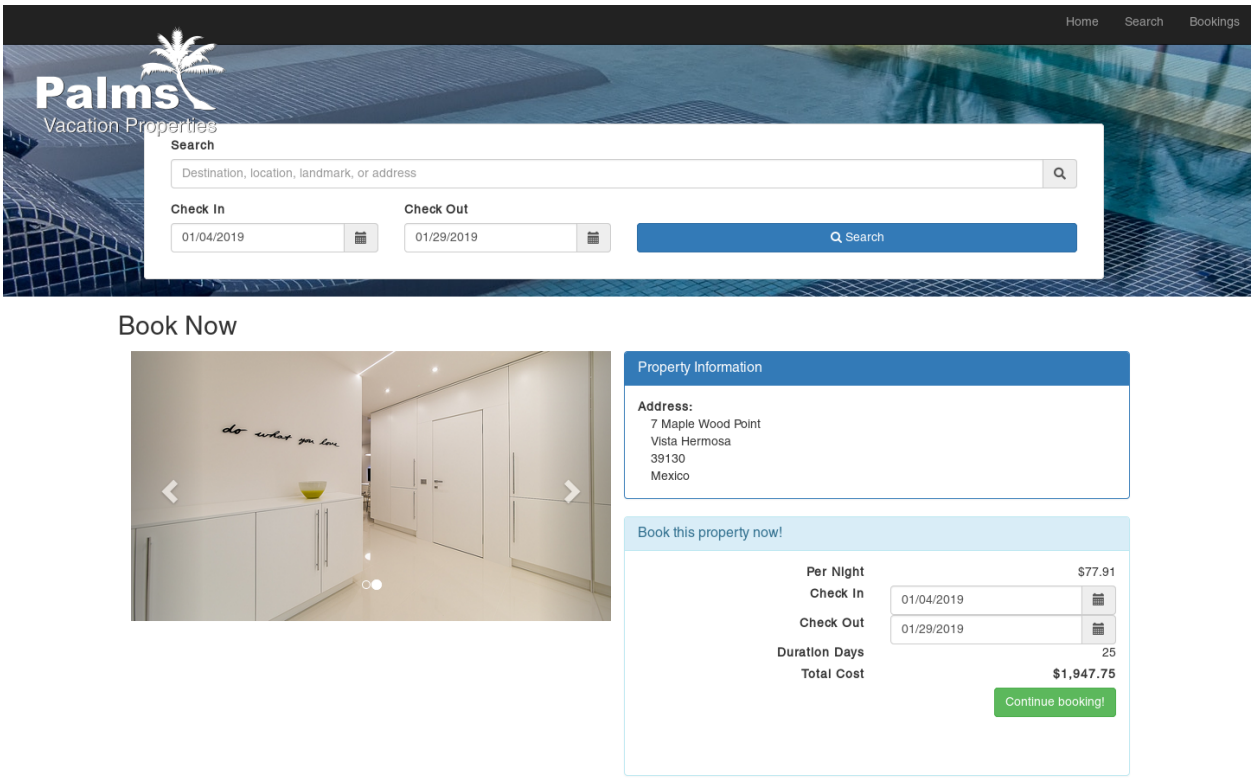
Main Navigation

The Header Menu bar and Header Menu are located in the horizontal menu bar. They are placed in this manner because generally people are accustomed to reading from the across the top of a page and find it easier to read from left to right across a screen. Also, we feauture the logo and name Palms Vacation Properties and a user can easily navigate to the home button. Moreover, the user is given a clean and minimal welcome page with the default and easy access to: Home, Search and Booking.



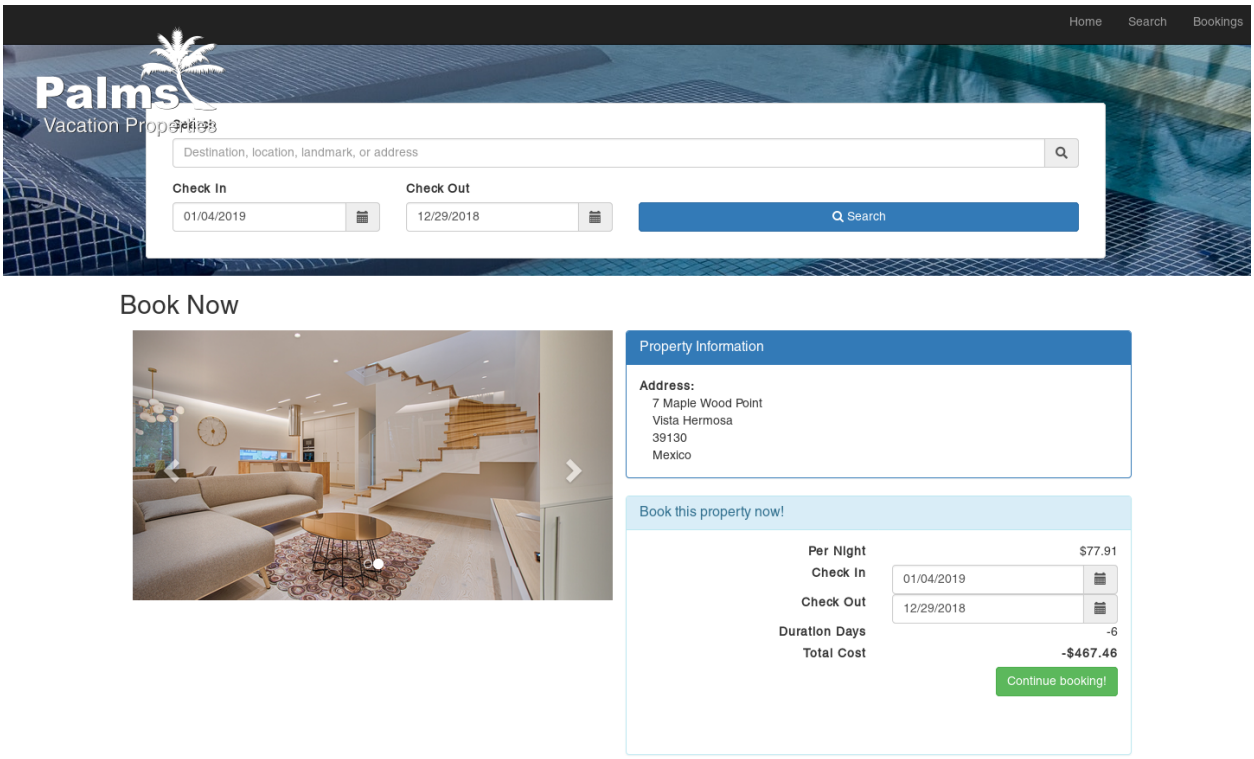
Search Box

The *search* box is one of the elements in our website that is a combination of input field and submit button. The search box is one of the elements that is most used in our website. We created a larger and more sizeable button in order for the user to click on it easier. The box is very simple and user friendly. The search box fits the overall design of the website style. Then, we have *check-in* and *check-out* buttons. The calendar in those buttons is formated as month/days/year(mm/dd/yyy). The checkin date is the first day of stay and the checkout date is the last day of stay. The length of stay is calculated from that.
Addittionally, we added rooms to be part of the search box in order to facilitate the search of number of rooms since we are dealing with vacation properties and not hotels. Then, we have size, a choice to choose spefic sizes that is measured in sq. ft. and so forth. Finally, we have prices-prices can be filtered according to the user's preference. If the user is open to any of the mentions filters then all the fields default to any and shows all the results that apply accordingly.



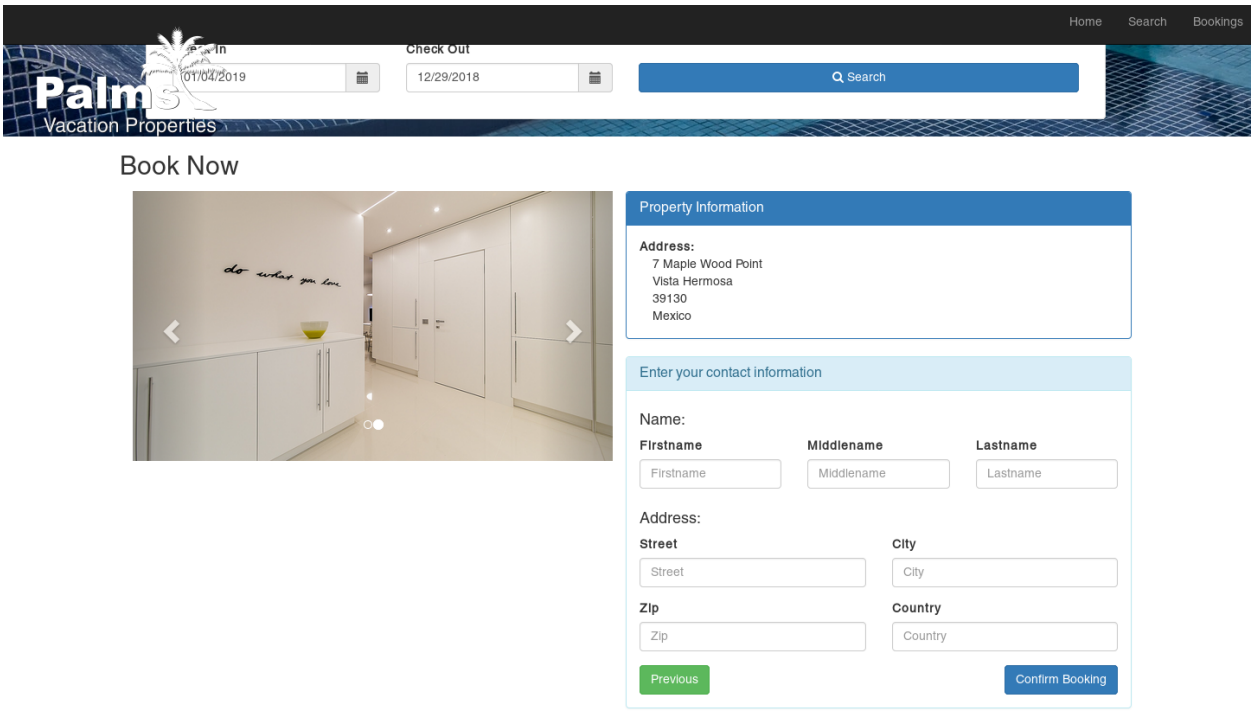
Book

When we click the *Book* button, we are able to begin creating a booking. Here, we will be able to see all the details of the property.



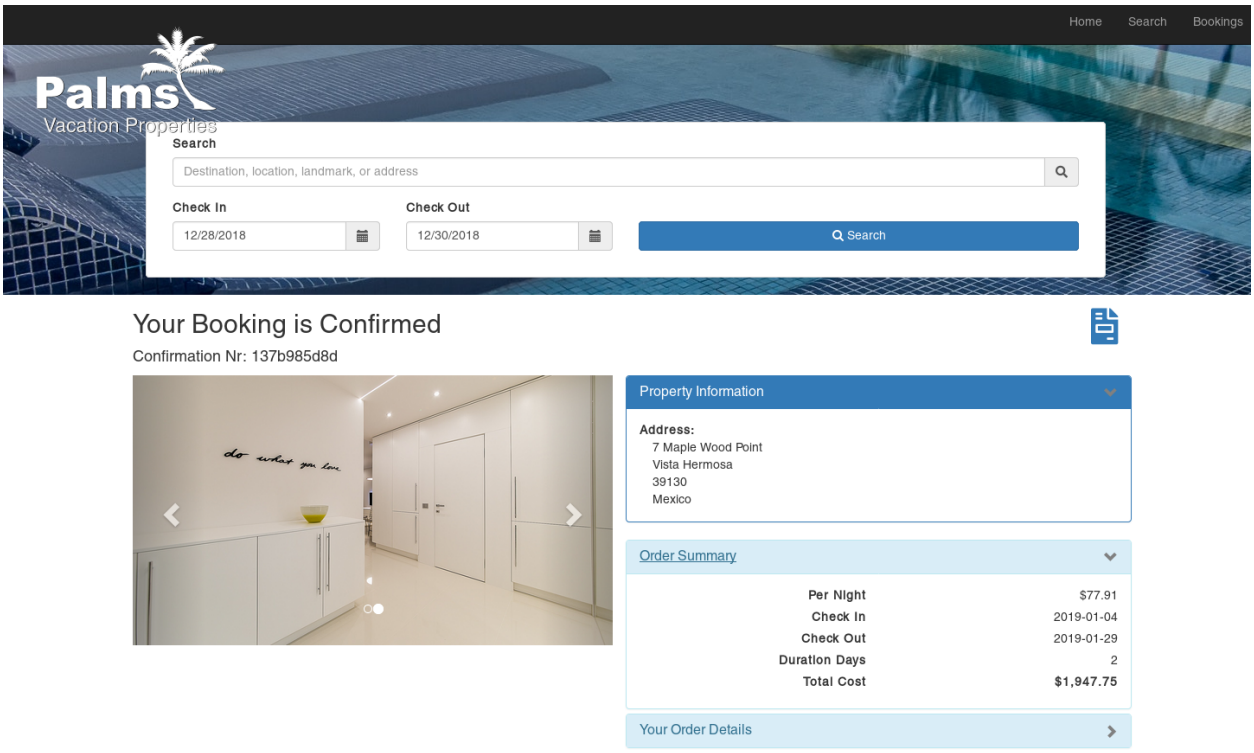
Book Now

When we click the *Book Now* button, we are directed property information such as address, the price per nigh, check in date, check out date, duration of stay, and total cost. Then, one can choose to continue booking.



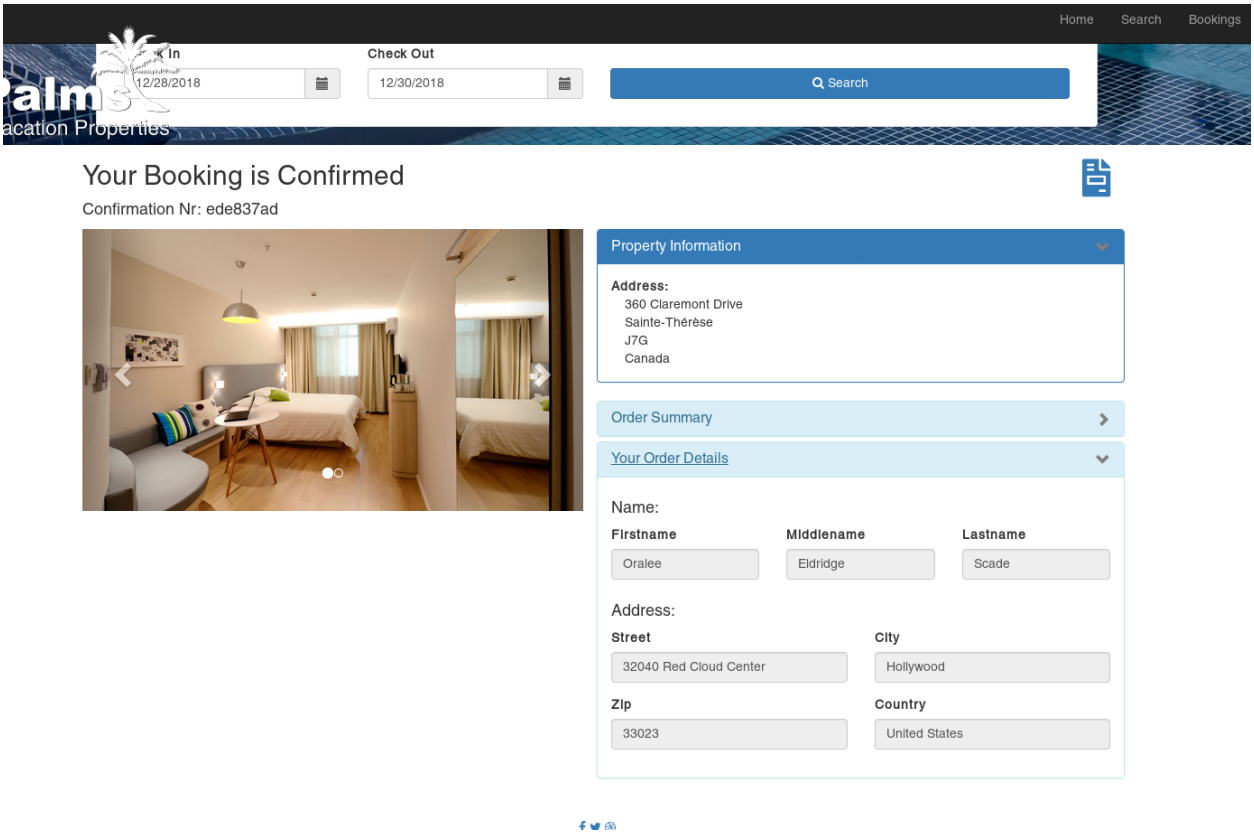
Continue Booking

When we click the *Continue Booking* button, we are directed to a page to enter guest contact information such as first name, middle name, last name, address and so forth. This section will then be redirected to Confirm Booking.



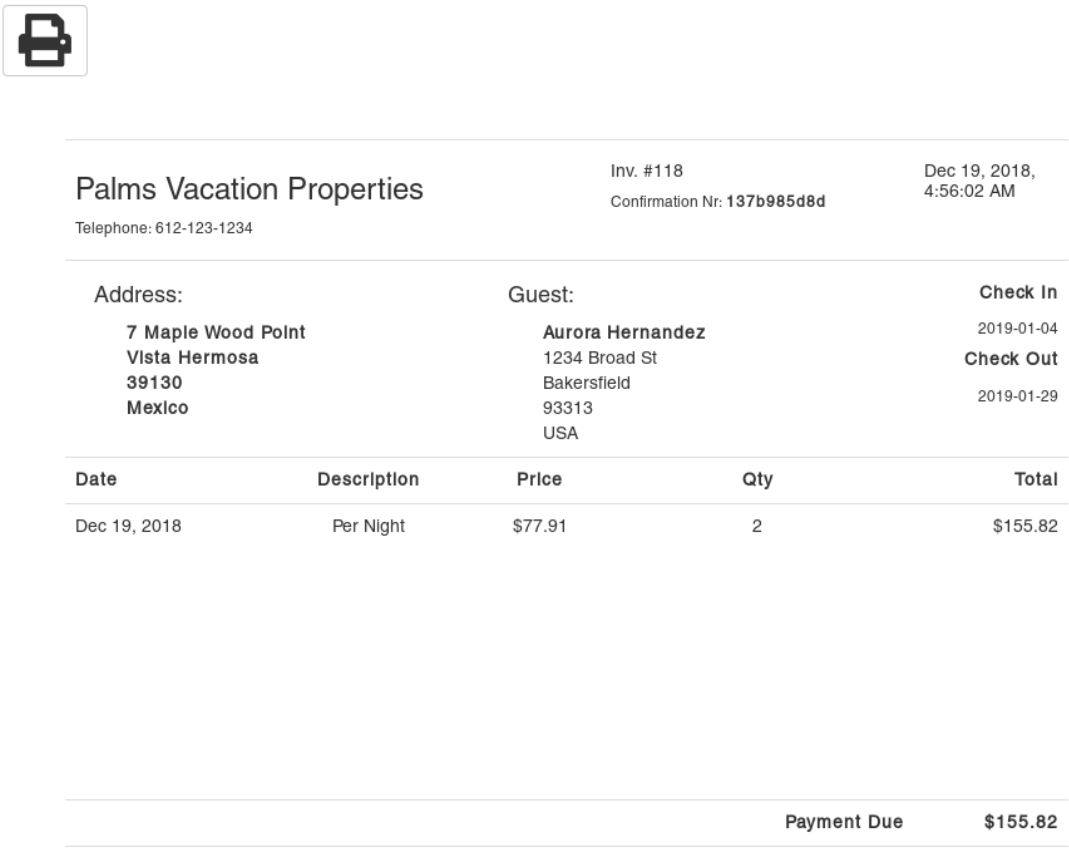
Confirm Booking

When we click the *Confirm Booking* button, we are directed to a page that tells the user that the booking is confirmed.



Booking Confirmation Details

After a user confirms their booking, they get all the confirmation details of their trip. A user is given a confirmation number and can further view their invoice and confirmation with further details.



Palms Vacation Properties

Telephone: 612-123-1234

Inv. #118

Confirmation Nr: **137b985d8d**

Dec 19, 2018, 4:56:02 AM

Address:

7 Maple Wood Point
Vista Hermosa
39130
Mexico

Guest:

Aurora Hernandez
1234 Broad St
Bakersfield
93313
USA

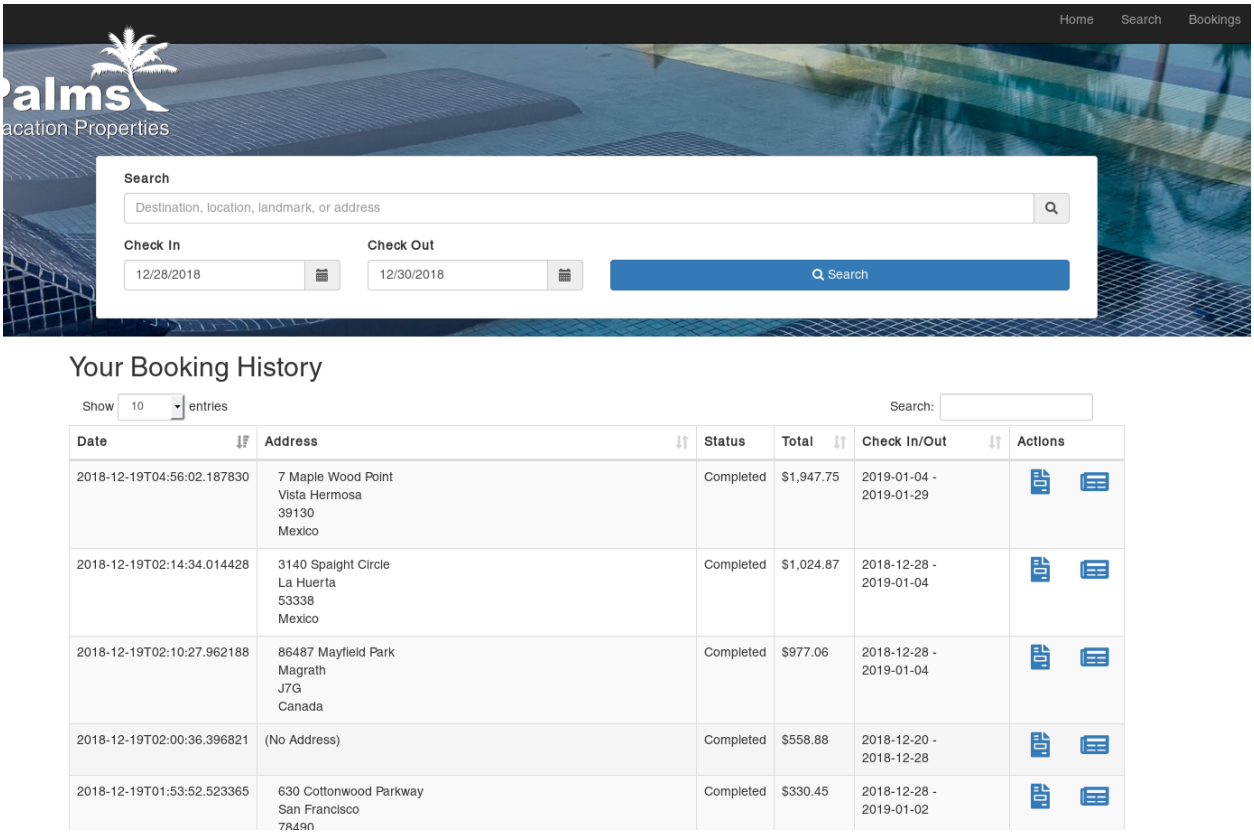
Check In
2019-01-04
Check Out
2019-01-29

Date	Description	Price	Qty	Total
Dec 19, 2018	Per Night	$77.91	2	$155.82

Payment Due **$155.82**

Booking Invoice

When we click the *Booking Invoice* button, we are redirected to a new page. Here, we can see all the details of an invoice. Then, a user can finally view the invoice or print the confirmation page with all the booking details and a PDF can be generated.



View All Bookings

When we click the *Booking* button, we are directed to a history of booking. Here, a user can see all the details of an invoice and a whole history of all the bookings a user has created. The booking history is organized with dates, addresses, status (i.e canceled, pending, void, completed and etc.), total amount, check in and check out dates. Then, one can finally view the invoice or print the confirmation page with all the booking details. A PDF can be generated if we click on the icon located under actions.

5.1.3 Tables, Views, Stored Subprograms, and Triggers

The application utilizes and provides all the required aspects for guests. The database takes advatantage of some of the PL/pgSQL functionalities offered through Postgress DBMS. However, our Palms Vacation Properties application specifically focused on the user interface of guests. There is a certain limit to our application, but it still uses most of the tables and views such as: Address, Guest, Booking, Property, Amenity, Attraction, View, Property_Rating, Phone, Includes, Selects, and so forth.

5.2 Programming Sections

This section embodies the programming techniques used for the implementation of Palms Vacation Properties application. In the next few sections, we discuss server-side programming techiques for each view and subprogram, we list the code and/or screenshot, and describe the purpose of the code. Next, we describe the middle-tier programming techiques that discloses code for database connection, code for sections which used view methods of calling stored subprograms. Then, we discuss client-side programming techiques for JavaScript code that is executed in the browser side.

5.2.1 Server-Side Programming

Server-Side Programming is a technique used in web development that involves scripts on a web server. This then produces a response customized to the user's request of the application. This technique is developed using WildFly; WildFly was formerly knowns as JBoss AS or JBoss. WildFly is written in Java and implementsin the Java Platform(Java EE) and runs in multiple platforms, its free and open-source software. By using this software, we take advantage of included EE APIs such as JPA.

To configure WildFly with a JDBC data source, the folowing script is run:

```
module add \
    --name=org.postgresql \
    --resources=/home/thor/Documents/CSUB/aurora_cmps3420/palms/web/pgjdbc-ng-0.6-complete.jar \
    --resource-delimiter=, \
    --dependencies=javax.api,javax.transaction.api,javax.xml.bind.api

/subsystem=datasources/jdbc-driver=postgresql:add( \
    driver-name=postgresql, \
    driver-module-name=org.postgresql, \
    driver-class-name=com.impossibl.postgres.jdbc.PGDriver, \
    driver-datasource-class-name=com.impossibl.postgres.jdbc.PGDataSource, \
    driver-xa-datasource-class-name=com.impossibl.postgres.jdbc.xa.PGXADataSource \
)

batch

/subsystem=datasources/data-source=palms:add( \
    jndi-name=java:/jdbc/palms, \
    driver-name=postgresql, \
    connection-url=jdbc:pgsql://localhost:5432/palms, \
    exception-sorter-class-name=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptionSorter, \

    ↪  valid-connection-checker-class-name=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker,
    ↪  \
    password=palms, \
    user-name=palms \
)

/subsystem=datasources/data-source=palms/connection-properties=database:add(value=palms)

run-batch

reload
```

Jboss WildFly DataSource and JDBC Configuration

This script first creates a Postgresql, which is then used to create a jdbc data source provider. Next, a jdbc datasource is created and a connection string used to connect to the Palms postgres database.

```
@Stateless
public class PropertyFacade extends AbstractFacade<Property> {

    ...

    @PersistenceContext
    private EntityManager em;

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    ...

    public PropertyFacade() {
        super(Property.class);
    }

}

```

Property Facade Service Class

In Java, a Persistence Context is injected into the service layer through configuration of the Application Server. This is transparent to the programmer as it is the Application Server's responsibility to manage the lifecycle of all data sources.

```
@Stateless
public class BookingFacade extends AbstractFacade<Booking> {

    public BookingFacade() {
        super(Property.class);
    }

    @PersistenceContext
    private EntityManager em;

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    ...

    void createBookingSp(Integer guestId, LocalDate checkin, LocalDate checkout, List<Integer> propertyIds) {
        getEntityManager().createStoredProcedureQuery("makeBooking")
        .setParameter(0, guestId)
        .setParameter(1, java.sql.Date.valueOf(checkin))
        .setParameter(2, java.sql.Date.valueOf(checkout))
        .setParameter(3, propertyIds)
        .execute();
    }

}

```

Make Booking Stored Procedure

This function calls the store precedure make_booking which takes in an guest id, check in and check out dates, as well as the properties that are being booked. The result is the booking id upon a successful transaction.

```
@Stateless
public class BookingFacade extends AbstractFacade<Booking> {

    ...

    //Returns a list of rows mapping to booking.* and property.*
    // outer joined on selecting property by id, ordered by pid
    List<Object[]> getAllBookingWithProperties() {
        return getEntityManager().createNativeQuery("select * from booking_property_info").getResultList();
    }

    ...

}

```

Get All From booking_property_info

Retrieve all rows from the view booking_property_info and return as a list of rows.

5.2.2 Middle-Tier Programming

The *Middle-Tier Programming* is the outermost layer and view layer that deals with presentation of the content and interaction of the user. In this tier, the application shows to the user the tools for interaction. This layer allows the user to inteact with the application. For example, the user can see different properties available and see different pictures.

For this implementation we used JavaServer Faces(JSF) which estrablishes the stardard-for building server-side user interfaces; JavaServer Faces is a Java specification for building user interfaces(UI) for web applications and part of Java Platform EE. We take advantage of the JSF APIs and tools to make our application smoother and easier.

Moreover, JavaServer Faces offers a separation between behavior and presentation for our application. The JavaServer Faces application maps HTTP requests to component-specific handling and manages components on the server. JavaServer Faces technology allows us to manage component state, processing component data, validating user input, and handling events.

5.2.3 Client-Side Programming

The *Client-Side Programming* has to do with user interface(UI) in web development. In the client-side, it is the program that run on the browser and deals with user interface. We deal with how the application interacts with temporary storage, work as an interface between user and server, send different requests to server, interaction with local storage, and handle interactive web pages.

In order to accomplish this, we use BootFaces. BootFaces is a lightweight JavaServer Faces(JSF) framework based on Bootstrap 3 and jQuery UI that lets you simplify the development of front-end applications. The features of BootFaces and BootStrap let us combine with JSF features to quickly develop our application. BootFaces also allows our application pages to automatically adapt on devices that range form mobile to desktop.

5.3 Survey Questions

This section deals with a few survey questions. We discuss grade on some personal achievement outcomes based on what was learned over the time of the course. The scale is between 1 - 10, 1 being the lowest score and 10 being the highest score.

(3b) An ability to analyze a problem, and identify and define the computing requirements and specifications appropriate to its solution.

9

(3e) An ability to design, implement and evaluate a computer-based system, process, component, or program to meet desired needs. An ability to understand the analysis, design, and implementation of a computerized solution to a real-life problem.

9

(3f) An ability to communicate effectively with a range of audiences. An ability to write a technical document such as a software specification white paper or a user manual.

9

(3j) An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices.

8